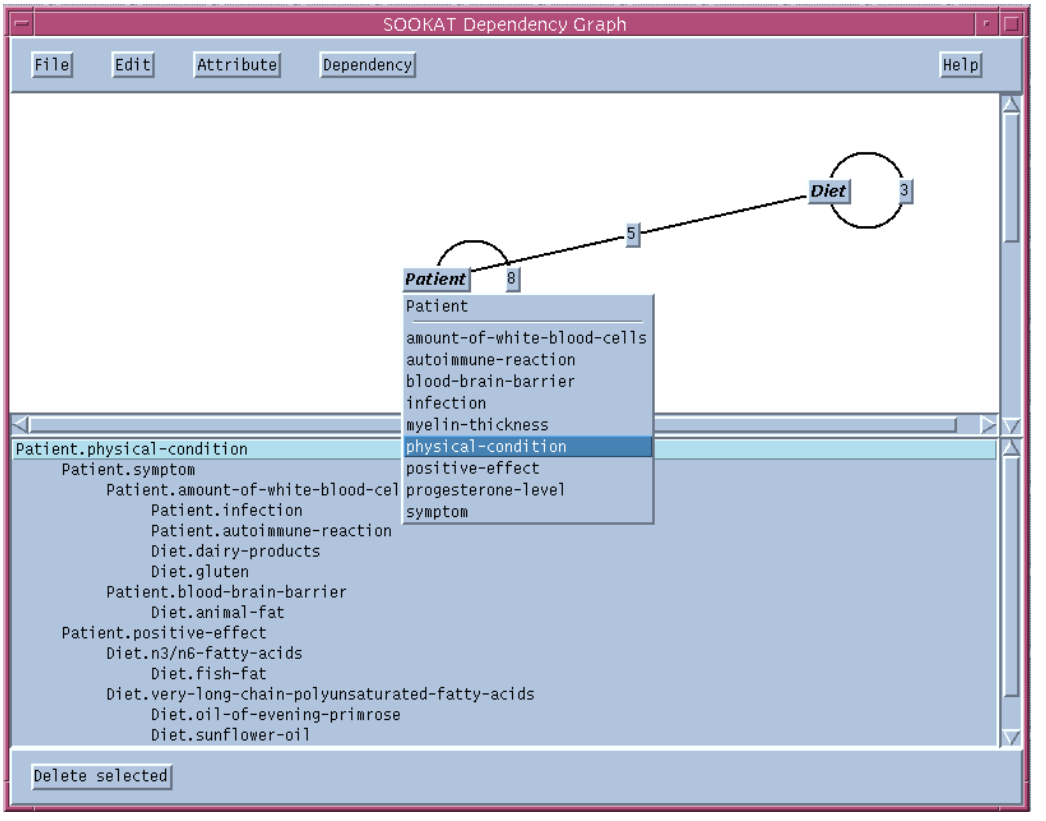
Department of Computer Science

Object-oriented knowledge acquisition

Integrating construction of and reasoning in object-oriented knowledge bases

Päivikki Parpola



Aalto University | SCIENCE + TECHNOLOGY | REPORT

Aalto University publication series
SCIENCE + **TECHNOLOGY** 13/2015

Object-oriented knowledge acquisition: Integrating construction of and reasoning in object-oriented knowledge bases

Päivikki Parpola

Aalto University
School of Science
Department of Computer Science

Aalto University publication series
SCIENCE + **TECHNOLOGY** 13/2015

ISBN 978-952-60-6460-4 (printed)
ISBN 978-952-60-6461-1 (pdf)
ISSN-L 1799-4896
ISSN 1799-4896 (printed)
ISSN 1799-490X (pdf)
http://urn.fi/URN:ISBN:978-952-60-6461-1

Unigrafia Oy
Helsinki 2015

Finland

441 697
Printed matter

Author
Päivikki Parpola

Name of the publication
Object-oriented knowledge acquisition: Integrating construction of and reasoning in object-oriented knowledge bases

Publisher School of Science

Unit Department of Computer Science

Series Aalto University publication series SCIENCE + TECHNOLOGY 13/2015

Field of research Knowledge acquisition

Abstract

Päivikki Parpola presents in this research report the SeSKA (seamless structured knowledge acquisition) methodology, integrating phases of knowledge acquisition (KA) through seamless transformations between object-oriented (OO) models. This attacks the problem of disintegration, or the gap between phases. The methodology is accompanied by presentation of the SOOKAT (structured object-oriented knowledge acquisition) tool supporting it. SeSKA and SOOKAT extend the KA process to constructing knowledge bases by instantiating a series of models for inferencing. The models are constructed in SOOKAT utilizing metaobject protocols.

Inferences performed in instantiations of OO models are guided by control objects (CO). Messages are sent between COs and components of the inference structure. A specific CO, possibly using subordinate COs, can be specified for each inference strategy.

There exists a mutual CO for forward and backward chaining that can also be used when reasoning according to protocols. In addition, COs for problem-solving methods (PSMs), such as cover-and-differentiate or propose-and-revise, can be used.

Three example applications are used for demonstrating the properties of the SeSKA methodology and SOOKAT, that is, a mineral classification "toy application", Sisyphus III rock classification and dietary management of multiple sclerosis.

Mechanisms for importing problem-solving methods (PSMs) over the Internet, as well as for generating specific control objects (COs) for them, remain open to further development.

Päivikki Parpola (1965-2015) was a Ph.D. student at Aalto University. Her research interests concerned knowledge acquisition and presentation, development and reasoning in expert systems for different application fields, using the object-oriented paradigm. She received her M.Sc. in 1988 and Lic.Phil. in 1995 from the Department of Computer Science at the University of Helsinki. Her M.Sc. thesis concerned forming a formal grammar based on text samples of natural language or unknown writing. Research presented in her Lic.Phil. thesis continued in her Ph.D. studies. She worked with Nokia Research Center from 1987 to 1993. In addition to her thesis, she published multiple international and domestic conference papers and articles as well as contributed in European Union research project publications.

Keywords Knowledge acquisition; modelling; object-orientation; metaobject protocol; knowledge management

ISBN (printed) 978-952-60-6460-4 **ISBN (pdf)** 978-952-60-6461-1

ISSN-L 1799-4896 **ISSN (printed)** 1799-4896 **ISSN (pdf)** 1799-490X

Location of publisher Helsinki **Location of printing** Helsinki **Year** 2015

Pages 18 + 120 **urn** http://urn.fi/URN:ISBN:978-952-60-6461-1

Contents

List of Figures

List of Tables

List of Examples

Preface

I met Päivikki Parpola first time in early 1990's, when she came to see me. She had received MSc degree from University of Helsinki, but wanted to continue her studies for the doctoral degree at Helsinki University of Technology. The reason for selecting me for the supervisor of her studies was (probably) the fact that the definition of the area of my professorship was "Computer Science, especially Knowledge Engineering" and her research interests were concentrated on "Knowledge Acquisition".

Frankly speaking the only connection was the word "Knowledge", because my research was focused on planning and scheduling. On the other hand there were at that time no professor of Computer Science in Finland studying Knowledge Acquisition, so I decided to help her. The early discussions between us where concentrated on how to transfer her credits from University of Helsinki to Helsinki University of Technology, how the requirements for her major and minor would look like and other practical aspects. Of course also her preliminary research plan was dealt with.

During these early meetings she was able to walk with a forearm crutch and her speech was normal. I, of course, noticed that she was disabled to a certain degree, but I did not know the reason why.

Next time I met her around the turn of millennium when she came to see me in a wheelchair accompanied by a personal assistant. She also had difficulties in speaking, but as we met several times I started to understand her speech better and better. I also learnt that she was suffering from MS (multiple sclerosis), a disease with no cure, only the advancement of the disease could be slowed down.

In order to understand the following we must have a quick look at how doctoral thesis are accepted in Finland. In theory the final decision is made by the university after a public (and oral) defense, where usually one (might be more than one) official opponent makes, sometimes difficult,

questions about the results presented in the thesis. Also the audience can make questions. The thesis is usually accepted, if not (which can in principle happen, but in practice happens very, very seldom) it is a scandal.

So in practice the real question is the preliminary reviewing, which formally is done in order to ensure that the thesis is worth of publishing (an official permit from the university to publish the thesis is needed before the public defense). There are usually two preliminary reviewers, the more well known on the field of study, the better. After receiving their statements the university can give the permission to publish the thesis, or not. This is by any means a formality, but can be ensured by selecting reviewers who are internationally well known, who are familiar with the contents of the thesis, and have a positive attitude to it.

From the scientific point of view the main thing at that time was that she had a definite research plan, which we could easily agree on. I did the best I could to help her, e.g. I financed her trip to an international conference in Nice. By the end of 2005 everything was in order: her manuscript, a monograph, was ready and sent to the language check and two well known researchers of the field in question, one from England and the other from Scotland, had given a preliminary agreement to act as preliminary reviewers of the thesis.

But Päivikki Parpola told me that she felt that she was not able, due to the state of her health, to defend her thesis in public and wanted to postpone the process until her health would be better. I could easily understand that point of view, but suggested that we could start the preliminary reviewing process, which most certainly would end in acceptance of the permission to publish, and wait for the public defense. Of course, I thought that the public defense could never take place, but thought that a manuscript with a permission to publish is better than nothing. But Päivikki Parpola did not agree.

Given the condition “until her health would be better” I thought that I shall never confront this problem again. In 2009 I retired from my duty as a professor of Helsinki University of Technology. Now five years after that and ten years after the original events, when everybody knows that public (and oral) defense is impossible for Päivikki Parpola, we have decided to publish her work through this channel in order to honor her excellent, by the standards of 2005, work.

The text above was written by January 21, 2015. The process stopped

when we learned, that Päivikki Parpola had passed away on February 10, 2015. Now, when the matter became again actual, I read the text above, correcting some mistakes, but I did not want to change anything. I hope, that those, who happen to read this study, would appreciate it.

Helsinki, September 2015,

Markku Syrjänen
Professor emeritus

Part I

INTRODUCTION TO KNOWLEDGE ACQUISITION AND METHODS USED IN IT

1. Disintegration in developing reasoning systems

The research field of developing *knowledge-based systems* (KBS), like expert systems, is called *knowledge acquisition* (KA). Disintegration means here the gap between phases of KA. It is a problem making KA more difficult, as described in section 2.3.1.

Disintegration has been overcome in KA tools built for specific purposes, using domain knowledge in all phases. In generic tools suitable for KA in multiple domains, other solutions must be found to solve the gap problem.

2. What is knowledge acquisition?

2.1 What is a knowledge base?

A knowledge base (KB) differs from a database in that in addition to containing data it also contains instructions of using the data. One possible implementation of this combination of data and use instructions is if-then rules. A KB using this implementation can be called a rule base. If a KB contains knowledge, normally possessed only by experts, it is called an *expert system*. A KB can be used as a basis for a larger system. Such a larger system is called a *knowledge-based system* (KBS).

2.2 Tasks of an expert system

An expert system is [Bratko, 1986], p. 314, a program used to

- solve problems requiring expert knowledge in some application domain,
- explain its behaviour and decisions to the user, and
- deal with uncertain completeness.

2.3 Building expert systems and other knowledge-based systems

The research field of developing KBS, like expert systems, is called *knowledge acquisition* (KA).

2.3.1 The gap between a model of knowledge and an executable knowledge base

Disintegration, or the gap between phases of development, especially between abstract and executable descriptions, has been recognised as a problem during early stages of KA [Marcus, 1988b, Motta et al., 1988]. It complicates the developent of KBs and hinders traceability between parts of abstract and executable descriptions.

The gap problem has been solved in narrow-focused automated tools [Marcus, 1988b, Leo et al., 1994, Grosso et al., 1999] that often use *problem-solving methods* (PSM), getting their power from understanding the roles that domain knowledge plays in problem solving. In problems that cannot use scope-restricting heuristics, the gap problem must be confronted using alternative solutions.

2.3.2 Collecting knowledge and transforming it into a form that can be executed

In order to construct KBS knowledge is elicited from domain experts or associated literature.

KBSs have traditionally been constructed by knowledge engineers (KE), who are people interviewing domain experts or studying expert literature, and formalising their knowledge. Different phases of KA require different skills of the KE [Klinker, 1989].

Computerised tools for KA aim at supporting, or even automating, the work of a KE. The ultimate goal is to construct more reliable expert systems in less time.

2.4 Tools and methods used in knowledge acquisition

Many KA tools support the building of an analysis model that then has to be manually coded into rules, or into some other presentation.

Tools developed for KA may differ a lot, as the ideas, experience and methods come from a variety of sources, and different phases of KA require different support. Different approaches form the hierarchy:

Automated tools The gap problem has been overcome in narrow-focused automated tools [Marcus, 1988b, Leo et al., 1994, Grosso et al., 1999] that often use *problem-solving methods* (PSM). These methods, getting their power from understanding the roles that domain knowl-

edge plays in problem solving, are described in chapter 3.

In problems that cannot use scope-restricting heuristics, the gap problem must be confronted using alternative solutions.

Machine learning techniques have also been used in some KA tools, but they will not be considered in this thesis.

Structuring of knowledge One generally applicable solution is structuring of knowledge. The original KADS methodology [Wielinga and Breuker, 1986] and the CommonKADS methodology [Schreiber et al., 1994] are described in chapter 4.

The structure of possible inference sequences, i.e. the *inference structure*, refers to components of the domain layer.

Structuring of knowledge eliminates the gap problem on domain and task layers. In reasoning, disintegration between abstract and executable descriptions may still present the gap problem, despite using inference structures.

Seamless transformations between models The transitions between models are seamless — we are ideally able to tell in a foreseeable way, how to get from objects in one model to objects in another [Jacobson et al., 1992]. Seamless transformations define the principles according to which a certain model can be constructed based on another model [Jacobson et al., 1992]. During KB construction, a series of models is created, some of which are modified during the development process.

Use of a metaobject protocol A metaobject protocol [Kiczales et al., 1991, Steele, 1990] contains objects on three levels of abstraction:

Metaclass is a class, the instances of which are themselves classes.

Class attributes with types describe a group of objects. Behaviour is defined through methods. Classes form an inheritance hierarchy. Classes can be created, modified and deleted while using a KA tool.

Instance is an instance of other class than a metaclass.

Protégé-2000 [Fridman Noy et al., 2000, Noy and Klein, 2004] uses a metaobject protocol to describe, for example, a model of CommonKADS [Schreiber et al., 2000]. This allows presentation of application knowledge as instantiations of the domain model.

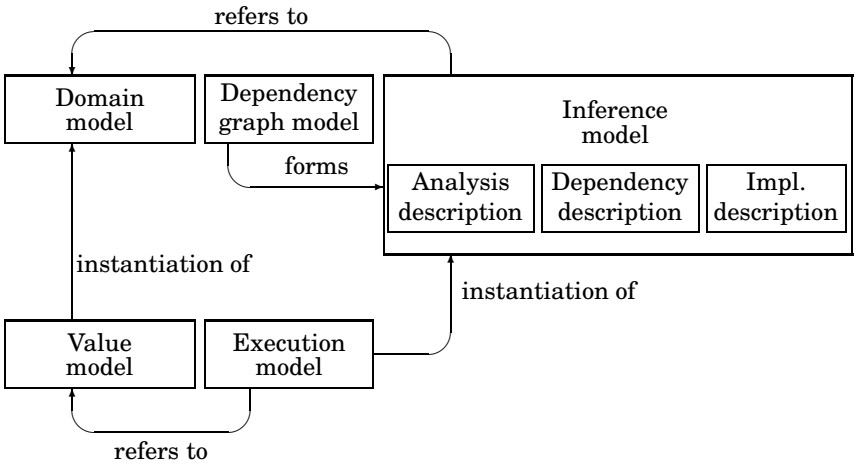


Figure 2.1. Models constructed in the SeSKA methodology.

2.5 The gap remaining in knowledge acquisition

The gap problem can be solved for concepts. In the general case, the gap problem still concerns descriptions of reasoning.

2.6 The SeSKA methodology and the SOOKAT tool

This thesis presents

- principles of KB construction, use, and maintenance according to the methodology *SeSKA* (seamless structured knowledge acquisition) [Parpola, 1998, Parpola, 1999b, Parpola, 1999a], as well as

- implementation of the ideas sketched in SeSKA, and reasoning according to a KB are done using the tool *SOOKAT* (structured object-oriented knowledge acquisition tool) [Parpola, 2005, Parpola, 2002, Parpola, 2001, Parpola, 2000].

SeSKA is developed to enhance integration of the KA process in several ways. During KB development, a series of models (figure 2.1) is created and modified.The structure of the KB is based on the logical structure of the domain which has been noticed to be more stable than the component structure [Jacobson et al., 1992].

All information in SOOKAT is presented and handled as objects. Use of

metaobject protocols in SeSKA and SOOKAT is described in sections 12.6 and 14.4.

3. Reusable problem-solving methods, and automated tools

In a few specific automated tools with narrow focus, the gap has been filled in. All these tools have a conceptual model operating on the knowledge level. Knowledge level describes the functionality of a KBS, i.e., *what* a system can do, opposed to symbol level description about the implementation, i.e. *how* does a system do its task. However, operating on the knowledge level does not mean that all tools with the conceptual model operating on the knowledge level would fill in the gap described above.

Preliminary steps towards a taxonomy of problem-solving methods (PSMs) have seen presented by J. McDermott in 1987 and 1988 [McDermott, 1988].

Musen [Musen, 1989a] classifies the conceptual models operating on the knowledge level into two classes. There are tools presupposing some problem-solving method, and tools focusing on tasks of a certain domain. Some examples of such tools are presented in sections 3.1 and 3.2.

A library of PSM skeletons is available on the Internet.

3.1 Method-based tools

Tools described in this section are MORE, MOLE, SALT, and S-SALT.

MORE [Kahn et al., 1985, Kahn, 1988] is a tool to develop KBs for solving diagnostic problems. MORE interrogates the expert in order to build a model composed of interrelated facts: events, symptomatic observations and their attributes, tests and states, events, and conditions strengthening or weakening hypothesis. The facts are represented in a relational network model. The model is used to generate rules, to which the expert must assign confidence factors to guide further interrogation of the expert and to recognise errors in confidence factors assigned to rules. MORE uses task-specific heuristics to accomplish all this.

MOLE [Eshelman et al., 1987, Eshelman, 1988] is a direct successor of

MORE. Like its predecessor, MOLE concentrates on heuristic classification tasks. It also uses a similar internal structure. The big difference between the two tools is in their user interface. MOLE uses domain-specific heuristics to be able to ask as little as possible in order to make the interview and use easier for the expert and for the end user.

Both MORE and MOLE construct sophisticated relational networks based on interviews they perform. The nature of diagnostic tasks makes it possible to produce rules that can be read directly from the internal network created, consisting of nodes that as such can be used as premises and conclusions.

SALT [Marcus, 1988b] is a tool generating expert systems for constructing designs, using the propose-and-revise strategy. It builds, based on interviewing, a dependency network with three kinds of relations: contributes-to, constrains, and suggests-revision-of. SALT then uses the internal representation to generate OPS5 rules [Forgy, 1981]. The functional knowledge base tells how and when the knowledge should be used during problem solving.

S-SALT is a successor of SALT, that can additionally refine its KB.

3.2 Task-based tools

Tools described in this section are OPAL and p-OPAL, the latter generated with PROTÉGÉ.

PROTÉGÉ [Musen et al., 1988, Musen, 1989b] is a meta tool for writing specific KA tools, supporting certain families of skeletal-plan refinement tasks. Using specific graphical forms, a knowledge engineer defines a task model and implicitly the conceptual model of the new domain-specific KA tool.

An example of a tool that can be created using PROTÉGÉ is p-OPAL [Musen et al., 1988, Musen, 1989b], that acquires new cancer-treatment plans. These are used by ONCOCIN, an expert system that provides cancer therapy advice (providing the inference engine and the UI).

A user of p-OPAL (a domain expert) enters knowledge in application-specific (medical) terms, using graphical forms:

- Instances and attribute values of predefined classes (planning entities, i.e., processes with a finite duration, e.g., chemotherapy, or task-level actions controlling the planning entities), and

- composition relationships among these instances.

The newly entered entities are stored in a frame structured intermediate knowledge base. This representation can later be translated by domain-specific programs into ONCOCIN knowledge bases:

- Flowchart diagrams for describing sequences of actions. These are automatically converted into augmented transition networks that ONCOCIN can interpret.

- Very specialised and limited kinds of rules that can have input data as conditions. The OPAL rules are translated into ONCOCIN rules that have added conditions corresponding to a specific context.

Having no gap between the modelling language and the implementation is in OPAL (and p-OPAL) achieved through careful adjustment to the domain and task at hand, sophisticated internal structures and processes, and sufficient limitation of the formalism.

Linster and Musen have created the conceptual model of ONCOCIN also using the KADS methodology [Linster, 1992].

3.3 Filling in the gap in the general case

As we have seen, certain domain and task specific tools integrate phases of KA. Their power comes from restricting the scope of the tool so that certain heuristics are applicable.

In the general case, only much weaker methods can be used. There has still been some effort towards bridging the gap between acquired knowledge and the implementation of a KB.

4. Structured knowledge acquisition — KADS and CommonKADS

KADS (denoting e.g. KBs analysis and design structured methodology; the acronym has been used to address several things in the course of time) is the best-known and probably the currently most developed methodology for structured development of KBs. The KADS methodology is described in its original sense [Hesketh and Barrett, 1989, Wielinga and Breuker, 1986] in section 4.1. CommonKADS [Schreiber et al., 1999] is a newer and better developed version of KADS. It is described in section 4.2.

4.1 KADS I

4.1.1 General

KADS [Hesketh and Barrett, 1989] is a methodology developed in ESPRIT project 1098 during the years 1985-1989. It covers the following areas:

- the process of KBS development,
- the life-cycle model for project management,
- support for activities involved in knowledge engineering, and
- tools and techniques for activities in KBS development. The original developers of KADS were B. Wielinga and J. Breuker [Wielinga and Breuker, 1986],

4.1.2 Analysis

Analysis is divided into three distinct streams, external, internal and modality streams, performed side-by-side.

External analysis External analysis produces a requirements model from external requirements that usually change. This model is a structured set of documents (including system, organisational and project requirements) that will not be considered until the design stage.

It is claimed [Heninger, 1980] that a software requirements document should satisfy the following conditions:

- It should only specify externally visible system behaviour.
- It should specify constraints on the implementation.
- It should be easy to change.
- It should serve as a reference tool for system maintenance.
- It should record forethought about the life-cycle of the system.
- It should characterise acceptable responses to undesirable events.

Internal analysis produces the *four-layer model of expertise* that is probably the most well-known and fully developed part of KADS.

The domain layer of the model consists of domain concepts, their attributes, and relations between them. The domain layer is task independent. The choice of the formalism sets constraints to what can be expressed.

The inference layer describes inferences, relevant to the task. Two abstraction issues are introduced:

- metaclasses: (Roles), i.e. domain groupings describing a role that a packet of domain knowledge may play in the inference process, not fixing a strategy or order for inferences,but rather describing a way domain relations can be used to make inferences, and

- knowledge sources (Inferences), representing discrete inference steps, or transformations of one or more metaclasses into a new metaclass.

If the same concept has several roles in inferencing, it is placed under several Roles. A typology of metaclasses (Roles) has been identified in KADS. The role a domain concept plays in inference is specified by the structure of Inferences, and the binding of concepts to metaclasses.

An *inference structure* is a network of knowledge sources (Inferences) and metaclasses (Roles), i.e. the description of possible inferences. It defines the constraints on the reasoning process, but does not give an ordering of inferences (no specification of how and when to perform them).

The task layer contains orderings of the inference structure. Depending on the task, there can be one or many such orderings. The inference structure is usually a set of rules of how to satisfy goals.

The strategy layer contains control knowledge for sequencing tasks. Often, the strategy layer is omitted as unnecessary.

Interpretation models (IM) are four-layer models without the domain layer, which the user has to define. These can be used to assist in developing new applications.

Modality analysis This phase produces the *model of co-operation*, which should contain the interactions that support co-operation between the final ES and its user. It includes two refinements on the task level structures:

- Assignment of ingredient (information, knowledge, skill) ownership and
- A specification of initiative agents (which agent may inititiate a task and thereby control the interactions that are required for its achievement).

The model of expertise and the model of co-operation together form the conceptual model.

There are two alternative ways to combine the model of co-operation with the model of expertise:

- Combine the task layer with the model of expertise. Ingredient transfer and monitoring tasks are inserted into the task structure.

- Use a meta-level co-operation manager. This separates the control of problem-solving actions from the higher-level goals of a particular session. (This approach is most suitable for complicated and/or flexible interactions.)

4.1.3 Design

The results of analysis can be used as the basis for the design. The structure of the conceptual model (model of expertise plus model of co-operation) is recommended to be preserved, so that there is a one-to-one mapping from the elements of the conceptual model to the elements of the implementation language. Use of, e.g., an intermediate language is recommended.

KADS design consists of three sequential phases, applying functional, behavioural, and structural viewpoints.

Functional design The functional architecture is designed in functional design. First generation KADS systems applied a two-layer architecture, placing domain and inference elements on one layer, and task and strategy elements on the other. In the second generation architecture, however, all four layers are separated. Proposed transformations [Schreiber et al., 1989] of the four layers of the model of expertise and relations between the different layers play an important role in constructing the functional architecture. KADS proposes that also the control of the problem solver and the communication are separated in domains with complex modality requirements.

Behavioural design Behavioural design in KADS means the selection of appropriate AI methods, that is, how the functional blocks determined in structural design are to be operationalized. Methods are defined by means of their data elements, i.e., data structures together with algorithms.

Physical design Design elements of the selected methods are composed into separate modules (the structural viewpoint), and an environment (e.g., Prolog or KEE) is selected. Exact definition of rules, methods or other representations is done. As KBSs are usually very complex, it is especially important to follow the basic principles of physical design:

- prevention of knowledge redundancy and inconsistency,
- uniformity of reasoning and representation, and
- modularity of the knowledge base.

Human-computer interaction (HCI) design The separation of communication control from problem solving allows the designer to focus on the user's perspective and the domain. Functional blocks for communication define the flows of information between the system and the user:

- what information,
- under whose initiative, and
- what kind of control is needed to achieve communication goals.

This constitutes a user-centred view of the domain.

KADS does not give especially detailed guidelines for HCI design. Environments supporting a wide variety of interaction methods are recommended in order to provide flexible use of KBs. Separation of HCI from problem solving makes it possible to pick the user interface implementation platforms among multiple choices.

4.1.4 Implementation

Implementation should ideally (even though often not in practice) be quite straightforward, as all the important decisions have already been made during design. These decisions include also exact definition of all the rules, methods, or other functional descriptions.

Development activities occurring after the implementation of a system are

- installation,
- use,
- maintenance, and
- decommissioning: making way for a new system.

An essential part of KADS is its spiral life-cycle model (handling mainly system development project management), which is applicable also in maintenance. A KB asymptotically approaches the desired performance. However, this requires much maintenance effort and a long period of time.

4.1.5 Tool support for KADS I

Shelley [Hesketh and Barrett, 1989], the tool supporting application of the original KADS methodology, provides rich support for building an analysis model. However, the transition from the analysis model to design and implementation is by no means direct, although drawing tools and guidelines for creating a functional decomposition are provided. In other words, a gap between the analysis phase and the other phases remains to be filled in by the knowledge engineer.

4.1.6 Critique of KADS I

KADS provides a very thorough framework for building a structured KBS. Aims and principles stated in KADS are strikingly similar to those in object-oriented software engineering [Jacobson et al., 1992], described in chapter 5. Both recommend structuring of the artefact produced, and maintaining the original structure, composed during analysis, throughout the life-cycle of the system in order to ease maintenance.

The KADS methodology, however, has a number of weaknesses:

Results only as text. Many of the phases, e.g., the requirements analysis, produce results only in textual form, which both makes the process heavy and produces an excessive amount of documentation.

Poor tool support for the design phase and especially for the transformation between the analysis and design phases. Transformations of domain-model layers to blocks of functional design have been proposed, though, and other guidelines have been presented.

What is still needed is a set of well-defined, seamless (two- directional) transformations, reaching up to the final code. It should, thus, be possible to transfer the work already done to the stage most appropriate for further development.

Handwork. KADS is considered by many people to be too heavy a methodology to apply in practical, industrial applications [Harmon, 1991]. In order to correct this, while still maintaining the structured approach, tools should be developed to automate everything that can be automated. It should also be considered what parts of the methodology are really needed.

4.2 CommonKADS

The CommonKADS methodology [Schreiber et al., 1994, de Hoog et al., 1994] provides a knowledge-based system (KBS) developer with a structured set of initial template models.

The CommonKADS model set consists of

- the organization model,
- the task model,
- the agent model,
- the expertise model,
- the communication model and
- the design model.

This set of model templates can be filled and modified during development. These templates have associated with them model states that characterize the landmark moments in development of models.

Project management can define activities that push models into certain desirable states. This way, management and the project work on the same set of templates.

Most of these aspects are outside of the scope of this thesis, so only the expertise model and the design model will be described.

There may be several expertise models, each modeling the competence of a certain agent (human expert, database, KBS, textbook, etc.). The expertise model consists of application knowledge and problem-solving knowledge. Both of these in turn consist of

- domain knowledge (concepts, relations and facts, and problem-solving methods),

- task knowledge (decomposition of the top-level reasoning task and division of control over subtasks, and competence theory), and

- inference knowledge (inference structure: structure of possible inferences performed on roles — labels for classes of domain knowledge, and strategic knowledge).

The links between these three categories are called knowledge roles.

Ontologies, i.e., meanings of different types of knowledge and their interaction, are defined on different levels, like

- the domain (concepts and relations),
- restrictions, or problem-solving methods, and
- the inference structure.

Mappings between different levels define viewpoints on the domain knowledge.

The design model serves as a bridge between the expertise model and the more operationality-oriented models. It consists of specification and design of the application, the architecture and the platform (language and implementation platform).

5. Seamless transformations in object-oriented software engineering

Ivar Jacobson and his colleagues [Jacobson et al., 1992] discuss in their book software engineering as an industrial process, and advantages of using objects in this process. They stress that in almost all systems requirements change unpredictably, and the systems have to be designed for incremental development over their life-cycle. Especially in large, constantly changing systems also reusability is necessary.

Object-orientedness can be characterised as a technique for system modelling. Using it, a system is modelled as a number of objects that interact with each other. An object-oriented (OO) model is usually easy to understand, as people think about the world in terms of objects. Another advantage of the OO approach is that modifications tend to be local.

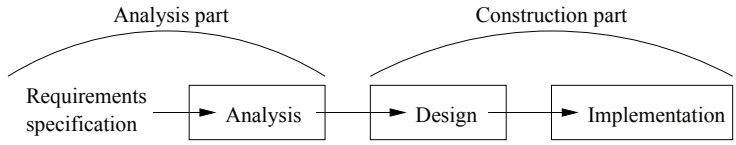


Figure 5.1. The analysis and construction parts of the OOSE process.

Object-oriented software engineering (OOSE), partially illustrated in Figure 5.1, consists of the iterative processes of analysis, construction, and testing. Requirements are used as input. During analysis, a requirements model and an analysis model are formed. The construction process produces a design model and an implementation model (the result code). A test model (including documentation, test specifications, and test results) is developed to support verification of the system.

The requirements model captures, right at the start, all the functional requirements of the system from a user perspective. In fact, three models are constructed — a use case model (the actual requirements model), an interface description, and a problem domain model. The analysis model

is based primarily on the use case model, not the problem domain model, as there is experience indicating that the logical structure is more stable than the domain structure. Thus, there will be less need for modifications.

When the requirements model becomes stable, the system is structured from a logical perspective into a structure that is robust and maintainable — the analysis model. It includes three kinds of objects — interface objects, entities, and controller objects. The transformation from the use cases into these objects is done by partitioning each use case according to certain principles. An ideal implementation environment is assumed at the analysis stage

In the design model, restrictions of reality are taken into account. Necessary decisions are made, and the application is further refined and formalised. The analysis model should be adopted with as little disturbance as possible, in order to make the system maintainable. The implementation model (the actual software) is gradually developed.

Design and implementation of the analysis model should be straightforward. The aim is to keep the logical structure of the analysis model in the final system. Thus, an important characteristic of object-orientedness (OO) is built-in traceability. An object identified during analysis must be found again in the code so that the system is easy to understand and durable to easy modifications.

The entire development process is actually model building. The system is gradually refined using the described models, or other ones. The approach supports modularity which induces locality of changes. Different parts of the system can simultaneously be at different stages of development. Some parts may be reused, possibly with slight modifications. By focusing on the more important aspects early, the base is laid so that the system structure is maintainable.

The transitions between models are seamless — we are ideally able to tell in a foreseeable way, how to get from objects in one model to objects in another model. This is absolutely crucial for an industrial development process, as the result must be repeatable. To be able to maintain the system it is also necessary to have traceability between the models. This will come as a side-effect of the seamless nature of the model transformations.

To summarise, the advantages of using the OO approach in software engineering are:

- The system is easier to understand.

- The system is easier to maintain.
- Changes tend to be local.
- Parts of the system can be reused.
- Components of the system can be developed separately.

6. What is a metaobject protocol?

6.1 Runtime metaobject protocols

A metaobject protocol enables accessing parts of an object system, such as classes and methods, as objects. A metaobject protocol can be implemented in two ways:

1. It can be *built in* to an object system. In these systems, the object classes utilised by a programmer are themselves instances of metaclasses, class and instance attributes are instances of attribute class, and so on. Built-in metaobject protocols are accessible as parts of programming languages.

2. It can be *built on top of* a programmable system, as an abstraction, possibly using an object-oriented programming language. In this case, classes, attributes, instances and other metaobjects are instances or data structures in the implementing system. As an example, this means that metaclasses are classes or data structure definitions in the object system and that the actual, manipulated classes are instantiations of them. Metaobject protocols built on top of systems are accessible as application programming interfaces (APIs) or user interfaces on add-on tools or libraries or object stores.

Regardless of their implementation, metaobject facilities in object systems can be divided into two categories according to their abilities:

(a) *Reflection* is the ability of an object system to provide the programmer with facilities to read and modify information about itself at runtime. The term reflection means that using these facilities at

runtime is immediately reflected to the object system. CLOS [Kiczales et al., 1991] and Smalltalk [Foote and Johnson, 1989] have built-in support for reflection.

(b) *Introspection* is reflection without actual abilities to modify the object system. Java [Sun Microsystems, Inc., 2001] and OpenC++ [Chiba, 1996] have built-in support for introspection.

6.2 Metaobject persistence

6.2.1 Built-in reflection systems

The simplest way to support persistence in a runtime object environment supporting reflection is by saving the system state as such, saving also any changes made to metaobjects.

- Smalltalk-80 does this by storing a system *snapshot* together with a *change set* [Goldberg, 1984].

- Different Common Lisp implementations usually have means for writing an *image* to save the system state.

These ways to save metaobject changes do not support reusability, portability or modularity. A Smalltalk-80 snapshot can only be taken into use by using the same *run* and *source files* that were used to create it. ANSI Smalltalk provides enhanced ways to overcome the problems, as described in section 6.2.2.

A Lisp image can be transferred and reused, but only in a system supporting the runtime environment. Also, any changes to the image are unique for it, since all the data in it shares the changes. Thus, sharing any changes between two Lisp images is very difficult, since an image with a modified object system can be considered unique.

6.2.2 Standard ways of storing metaobjects

The following ways are considerable for making metaobjects persistent:

- ANSI Smalltalk [X3J20 Workgroup, 1998] defines a *Smalltalk Interchange*

File (SIF), a standard that supports saving code from Smalltalk, to be read into Smalltalk. This enables portability and interoperability: All standard Smalltalk implementations can read and write SIF-conforming code. ANSI Smalltalk also supports the concept of a *package* to a SIF, solving the modularity problem. This is a programming language-specific solution. It manages both metaobject data and functions, i.e., methods.

- Meta Object Facility (MOF) [Object Management Group, Inc., 2002] by OMG supports metamodels with
  - classes,
  - associations,
  - data types and
  - packages.

  XML metadata interchange (XMI) [Object Management Group, Inc., 2003] is a specification for handling e.g. MOF metamodels as XML datasets. Used together, a metaobject system and means for storing and delivering that to other systems can be created. Some implementations exist. This solution depends neither on the programming language nor the computing environment. Representing object functionality is not part of the specification.

- RDF Schema [Brickley and Guha, 2004] and OWL Web Ontology Language [McGuinness and van Harmelen, 2004] are languages for processing Web content information. The former is a vocabulary for describing classes and properties of Resource Description Framework (RDF) resources and the latter additional resources for that to describe ontologies. Both are World Wide Web Consortium (W3C) semantic Web recommendations. These solutions are platform-independent, describing ontologies without functionality. Numerous implementations exist.

6.2.3 Metaobject managers

Special tools have been built for providing facilities for managing KBs. Expressing knowledge as objects has lead to the need to manipulate and store metaobjects, especially classes. These systems often have their origin in knowledge technology and/or are implemented to support software development. ConceptBase [Jarke et al., 2002] is a deductive object man-

ager for knowledge-based systems. It supports metaclasses and rules. MARVEL [Kaiser, 1993] is a knowledge engineering and a software development environment supporting modifying a data schema at runtime, while objects instantiating the schema remain consistent.

6.2.4 Proprietary solutions for specific purposes

When an application needs to define metaobjects for a special need, possibly setting some special requirements for the object system, a feasible solution is to have a metaobject protocol built on top of a programmable system.

Metaobject persistence in this case can be achieved by storing ordinary object instances. Choice of the solution for persistence may vary according to

- object model complexity,
- speed complexity requirements,
- space complexity requirements,
- interoperability requirements, and
- portability requirements.

With Java, the selection may be

- serialisation,
- any extensible markup language (XML) application programmer's interface (API),
- resource description framework (RDF) API,
- Java data objects (JDO) [Java Data Objects Expert Group, 2003],
- Java database connectivity (JDBC), or
- some proprietary alternative.

Part II

EXAMPLES REFERRED TO IN THIS THESIS

7. Mineral classification toy application

7.1 Background

The purpose of the mineral classification toy application is to provide an example as simple as possible to facilitate in demonstrating the properties of the SeSKA methodology and SOOKAT. The term “toy application” stems from the small sizes of the domain and the KB. However, they suffice for presenting, how incomplete information is handled and objects are used, both in a KB and in reasoning. The terminology in this example application has been borrowed from the Sisyphus III example in chapter 8 to reflect different models in the application to a real domain.

7.2 About the domain

The domain consists of two minerals, Olivine and Pyroxene, samples of which should be differentiated. The properties considered are colour and crystal shape. The colour of both minerals is green. The crystal shape of Olivine is octagonal, whereas that of Pyroxene is tabular.

7.3 Problem attacked

The problem is to determine the mineral type of a sample mineral. In other words, the task is to tell whether the sample is an instance of Olivine or Pyroxene.

8. Sisyphus III rock classification

8.1 Background

The Sisyphus experiments [Shadbolt et al., 1996] are attempts to compare and evaluate different knowledge acquisition (KA) methods and techniques. The Sisyphus I project (the room allocation problem) that began in 1990, provided much insight, but was criticized to be too simple. Sisyphus II dealt with design of complex mechanical devices, using the domain of an expert system called VT [Marcus, 1988b], that was built to design lifts. Results of Sisyphus II can be divided to

- more insight and lessons learnt,
- tools, that can be accessed from the Web, and
- standard model, containing concept features mentioned in transcripts and ontologies collected, that different approaches and participants can contribute to.

Later, newer techniques have been used. Sisyphus III, described in the next section, concerns rock classication based on rock samples, features of which are mentioned in interview transcripts, originally published on the Web site of the University of Nottingham. Transcripts are codings of expert interviews. Inference rules are given in transcripts, that reason the category of a rock sample. Sisyphus IV, proposed in KAW '96, dealt with using the Web as a KA tool. The Sisyphus V problem is to produce a Web version of Sisyphus I.

8.2 About the project

The Sisyphus III project was onset after the Banff KA workshop in 1995. Its principal objectives are:

1. to provide quantitative comparison of systems and methodologies used in them, through a set of achievement metrics,

2. to provide more realistic access to actual KA material in a staged series of releases, and

3. to obtain more complete records, i.e. knowledge engineering meta protocols, concerning the processes that the knowledge engineer goes through in the KB construction process.

The scenario for the Sisyphus III project is to construct a geology KBS for rock sample characterisation. The system is ultimately intended to act as a training aid and Decision Support System (DSS) for trainee astronauts. The system should help the user to provide an appropriate description of the sample type and support discrimination of the initial set of 16 igneous rocks. The system is to be used in conjunction with hand specimen, a hand lens (a geology loupe with a handle) and thin sections (slices of specimen mounted on a microscope slide for examination).

Criteria for comparison of systems and methodologies, i.e. the measure used are:

- efficiency — time and effort spent to produce a system of a certain fidelity

- accuracy — the discriminatory and explanatory power of the built system

- completeness — the coverage of the system with respect to certain fixed points in the domain

- adaptability — the ability to extend the problem solving functionality of the system

- reusability — the ability to reuse elements of the system in either extending the domain coverage or the problem solving functionality

- traceability — the ability to relate elements of the final system behaviour to its provenance in original KA material

8.3 Problem attacked

The task is to build an expert system, using which an astronaut, i.e. a person with no expertise in mineralogy, can determine, based on feature values of a rock sample, which rock category of the alternatives described below the sample belongs to. These feature values must be available through examination with a hand lens or a simple microscope.

Some of the rock types are recognized based on feature values. Some of the feature values of different rocks or minerals are explicitly mentioned in transcripts of interviewing multiple experts according to several interview strategies . In some cases, all experts, or majority of the experts, suggest the same values. Some feature values, however, have to be reasoned based on values of other features,by using rules, explained in the interviews. These rules may have to be applied several times.

During the analysis process a number of features have appeared to be significant in identifying rock types. Equally, it has appeared that minerals can be characterized using certain features.

Interview transcripts have been gone through to get value opinions for different features of different rock and mineral types, given by different experts. These have been analyzed in order to get default values for different features of every rock type. Also rules have been elicited and applied for inferring certain features of rock types based on

- other features of the same rock types, or
- features of compositionally similar rock types.

Expert statements, as well as rules have been utilized. All this has been performed in several rounds, so that different feature values get reasoned.

Example 1. *The rock type gabbro, the grainsize of which is coarse-grained, is compositionally similar to both dolerite, that is medium-grained, and basalt, that is fine-grained. Also other compositioonal similarities can be*

found from expert interviews. E.g. large- i.e. coarse-grained diorite is reported to be compositionally similar to both trachyte and andesite, that are both small- i.e. fine-grained.

Expert statements, as well as rules, have been utilized. All this has been performed in several rounds, so that different feature values get reasoned.

Example 2. *Both rocks and minerals are implemented as instances of metaclass Concept, that is, as subclasses of application classes* Rock *and* Mineral. *Minerals included in rocks can be described through aggregate attributes. Concept instances know about their subclasses and superclass.*

Instances of application classes Rock and Mineral and their subclasses, that are all instances of metaclass Concept, are implemented in the value model as instances of class ConceptInstance. Instances rock#21 *and* mineral#23 *of Rock and Mineral override their attributes with new attributes having values, corresponding to their appearance.*

9. Dietary management of MS

9.1 About the domain

Multiple sclerosis is a disease of the central nervous system, causing malfunctions mainly in the motor and tactile nerves, as well as in optic nerves. The symptoms are caused by autoimmune reactions against the protective myelin sheath of nerves.

MS is not an inherent disease, but is caused through combinations of groups of genes and environmental factors.

9.2 Problem attacked

A KB is being designed for helping a person with multiple sclerosis (MS) develop an optimal individual diet. This diet-planning KB takes into account:

- how nutrients affect MS in general,
- what nutrients individuals are sensitive to (what allergies etc. they have), and
- mental attitudes – how much individuals are ready to change their diet. Dietary consulting [Fitzgerald et al., 1987] may affect these attitudes.

The KB gives literature references in its explanations, so that the importance of nutritional recommendations, and possible need for changes in diet, can be evaluated.

9.3 How nutrients affect MS in general

Research concerning effects of nutrition on MS has concentrated mainly on fatty acid levels [Bates, 1990, Cunnane et al., 1989, Eshelman et al., 1987, Marshall, 1991, Neu, 1985, Fitzgerald et al., 1987, Navarro and Segura, 1989], intake, and absorption, as well as effects [Gallai et al., 1995, Mayer, 1991, Nightingale et al., 1990, Navarro and Segura, 1989].

What is essential about fatty acids is their saturation [Swank, 1991]. No more than 10-15 per cent of intake of fats should be saturated. Milk fat as well as coconut and palm oil are mostly saturated [KTL Nutrition Unit, 2004]. Also a remarkable part of fat of mammal animal meat (red meat) is saturated [KTL Nutrition Unit, 2004]. However, also meat containing less saturated fats exists, and meat is a good source of vitamin B12 [KTL Nutrition Unit, 2004],that is important in MS [Miller et al., 2005, Reynolds et al., 1992]. In practice, the only sources of vitamin B12 in nutrition are animal originated, including meat of mammal animals, like pigs or cows,meat of birds like chicken or turkey, or meat of certain fish species, like salmon, or milk of e.g. cow or goat.

This alone poses a challenge to a person with MS trying to avoid saturated fats. Another challenge is a possible vitamin B12 absorption disorder, possibly causing low or decreased levels of vitamin B12 demonstrated in people with MS. The challenges above can be met by supplementing a diet with vitamin B12 as pills or as intramuscular injections in the case of a demonstrated absorption disorder.

Fish oils and certain vegetable seed oils are a good source of unsaturated fats. Unsaturated fatty acids belonging to the Ω3 and Ω6 groups are based on the essential fatty acids linoleic acid and linolenic acid, that human body cannot itself produce [Food and Nutrition Board and Institute of Medicine, 2002], so they must be acquired through diet.

Dietary sources of essential fatty acids include the following [KTL Nutrition Unit, 2004, Hyvönen and Koivistoinen, 1994]:

Ω3: flaxseed i.e. linseed oil,
rape-seed oil,
soy oil, and
fish fat;

Ω6: sunflower oil,
corn oil,
soy oil, and
rape-seed oil.

Most (perhaps all) fats, even vegetable seed oils, are partially saturated. E.g. about 10% of fatty acids in sunflower oil are saturated.

Figure 9.1 illustrates this hierarchy of fatty acids. MS seems to be associated with insufficient unsaturated fatty acid intake: A typical diet resulted in a 6% intake of calories from unsaturated fatty acids in an MS group vs. a 17% intake in a control group [Azadbakht et al., 2002]. People with MS need essential fatty acids even more than healthy people, due to metabolic malfunctions in the intake and utilization [Zamaria, 2004].

Also effects of selenium [Lindemann et al., 2000, Clausen et al., 1988] and other antioxidants [Clausen et al., 1988], different vitamins, and lactic acid, as well as flavonoids have been investigated. Lactic acid makes recovering from fatigue, e.g. after physical straining, faster.

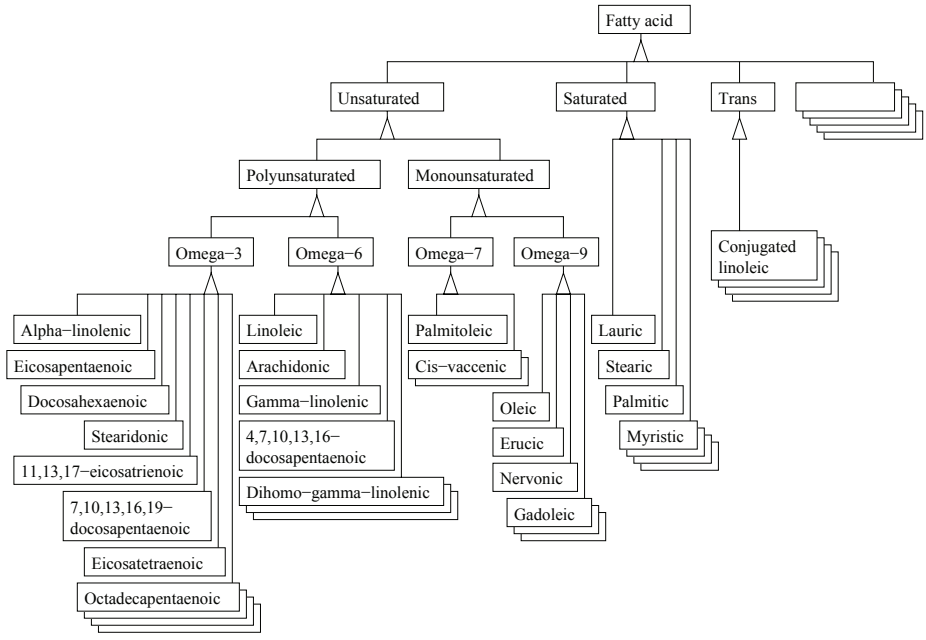


Figure 9.1. Classification of fatty acids in nutrition, presented as an inheritance hierarchy.

The newest research results concerning MS stress the importance of vitamin D. Effects of vitamin D are twofold:

- it lowers the incidence of MS in statistical analysis [Munger et al., 2004, van der Mei et al., 2003], and

- it lowers risk of getting consequences like

  – increased bone resorption,
  – fractures, and
  – muscle weakness [VanAmerongen et al., 2004, Kalueff et al., 2004].

People with MS should get double the amount of vitamin D recommended for healthy people [Munger et al., 2004].

Knowing about beneficial and unbeneficial components of different nutrients, unbeneficial ingredients can especially in cooking, be replaced with healthier ones.

To summarize, people with MS need supplementation (that may be also dietary) in:

- unsaturated fatty acids belonging to the Ω3 and Ω6 groups,
- vitamin B12,
- vitamin D,
- antioxidants,
- lactic acid, as well as
- fresh fruit and vegetables.

9.4 Planning the support for dietary management of MS

Dependencies are formed to represent research results concerning dietary management of MS. These dependencies are combined to form the initial DG, simultaneously with forming the DM. This is presented in detail in section 15.3.

Part III

SEAMLESS STRUCTURED KNOWLEDGE ACQUISITION

10. Strategies used against disintegration

Section 2.4 describes tools and methods used in constructing KBs. Three of them contribute to KA strategies that make integration possible in the general case, that is,

- use of a metaobject protocol to enable using object-oriented models for a suitable *uniform representation formalism*,
- structuring of knowledge to *structuring a KB* and
- seamless transformations between models to *seamless transformations between different KA phases*.

These strategies will be shortly described in this chapter. Chapter 11 describes integration of these strategies in the KB development process, enhanced by a suitable structure.

10.1 Use of a uniform representation formalism — the object-oriented paradigm

A major cause of disintegration between different phases of KA is apparently the different nature of the phases, leading to different representations. Observation-oriented gathering of knowledge normally cannot be directly utilized in development. Use of different, or incompatible, representation formalisms explains also other forms of disintegration.

There are more than one possible representation formalisms. Choosing compatible ones to be used can be done by choosing one formalism. In this work, the OO paradigm is proposed to be used as a unifying formalism, used throughout the development process. The OO paradigm distributes

representation over a number of active entities, called objects, defined to represent abstract or concrete concepts.

10.2 Structuring

Representation formalism alone is not powerful enough to integrate models, but also suitable structuring has to be used. The basic idea of separating domain and inference models is adopted from the methodology CommonKADS [Schreiber et al., 1994], performing structured KA as described in Chapter 4. Ontologies, presenting acquired knowledge in the form of dependencies between attributes of domain concepts, are augmented to the basic structure.

10.3 Seamless transformations

The effect of representation and structuring can still be completed using seamless transformations between different models. The idea comes from software engineering. A methodology called OOSE (Object-Oriented Software Engineering) [Jacobson et al., 1992] defines seamless transformations between object-oriented models, corresponding to different phases of development. Seamless transformations give predefined ways of getting from one OO model to another.

11. Representation structure enhancing integration

Integration of models depends, in the framework presented in this thesis, much on the representation structure. This consists of three main models presented using OO networks. The models used, containing the components presented in parenthesis, are

- the domain structure model (concepts with attributes, and relations),
- the dependency graph (attributes of concepts, and dependencies with descriptions), and
- the inference structure (roles, and inferences with analysis, design and implementation level descriptions),

11.1 Domain structure model

A *domain model* (DM) is a description of relevant parts of the domain. Relevant means here "involved in the task at hand". A DM consists of *concepts, with attributes* describing (the state of) different features, and *relations*, describing stable (structural) relationships between concepts or attributes of concepts. Relations between concepts include inheritance relations.

Tools like KEATS [Motta et al., 1988] help in forming the domain structure model from text.

11.2 Inferential dependency model

According to Jacobson and his colleagues [Jacobson et al., 1992], the *logical structure seems to be more stable than the domain structure*. The dependency model first initially acquired, and later updated, is a dependency graph (DG), consisting of attributes of domain concepts and binary, directed dependency relations (dependencies). The (inferential) dependencies described must be true in some situations, but they need not necessarily be always valid. Dependencies in a DG can be of one or several types.

Untyped DGs have been used in a system called Matias [Kontio, 1991]. The automated tools MOLE [Eshelman, 1988] and S-SALT [Marcus, 1988b] use untyped and typed dependencies between events.

11.3 Inference model

An *inference model* (IM) is a combination of

- an *inference structure* (IS) and
- descriptions of all inference steps,

forming together the analysis, design, and implementation models.

11.3.1 Inference structure

Knowledge acquisition as pure knowledge base construction concentrates on trying to translate acquired descriptions of inference to a computerized form. Following the example of KADS [Wielinga and Breuker, 1986] and CommonKADS [Schreiber et al., 1994], with minimal modification, the initial KB contains an *inference structure* (IS), consisting of *roles i.e. groups of concept attributes* and *inference steps between these roles*. The first informal (analysis) descriptions of inferences is formed based on the dependency graph.

Quite detailed knowledge (in fact, all details) can be presented already in the analysis descriptions of inferences. This means, that it is safe to fix, already at the analysis stage, a common inference structure, used also by design and implementation. Executable modelling languages [Lukose, 1995] are examples of presenting detailed knowledge in early descriptions of

KA. Integration of inference structures of course requires, that representation formalisms, used in different descriptions of inference, permit such integration. If so, the effort required in developing the design and implementation descriptions is to formalize the analysis descriptions of inferences. Changes (including changes in the inference structure) can always be made first to the analysis descriptions, and then be propagated to other descriptions.

When using *object-oriented representation*, integration of inference structures, used during different stages of development, is possible in the way described above.

Components of the inference structure, i.e. roles and inferences, are each presented by an object class. Descriptions of inferences, produced during different phases of KA, e.g., *text produced during analysis*, and *rules or functions produced during implementation*, are all *attached to the same inference object as attributes*. Using the same inference structure guarantees traceability between corresponding parts of different descriptions. It also fulfills the definition of seamless transformations, that can be used as a tool for bridging the gap between different phases of development.: Sharing of inference structures provides a basis for semi- automatic integration of KA phases.

11.3.2 Analysis, design and implementation descriptions

The inference steps in an IS have attached analysis, design, and implementation descriptions, i.e. the major logical components of abstract descriptions, their formal descriptions, and executable rules or functions, respectively. Together, they describe the structure of possible inference sequences through a network of roles and inference steps.

Analysis, design and implementation descriptions are formed iteratively. However, analysis descriptions are the starting point. Design descriptions are formalized from the analysis descriptions, in order to analyse them and to resolve any problems or contradictions. Implementation descriptions, i.e., rules or possibly functions, are generated manually or automatically from analysis or design descriptions.

Combined with the inference structure, the different descriptions form the analysis, design and implementation models.

12. Essential features of SeSKA

12.1 General

Seamless structured knowledge acquisition (SeSKA) [Parpola, 1998, Parpola, 1999b, Parpola, 1999a] is a methodology for the development and maintenance of KBs. It is designed to enhance integration of the KA process.

During KB construction, a series of models (figure 12.1) is created, a number of which are modified during the development process.

The structure of the KB is based on the logical presentation of the domain which has been noticed to be more stable than the domain structure [Jacobson et al., 1992].

12.2 Use of uniform formalisms

A series of models with varying characteristics, created in different KA phases, should use compatible formalisms in the phases. The choice is made between conceptual graphs (CG) [Sowa, 1984] and the object-oriented (OO) paradigm [Parpola, 1998], choosing either to be used in all models. In SeSKA, the OO paradigm has been chosen as the presentation for the models. It is used in initial presentations of the acquired domain knowledge, both for somewhat stable knowledge in the DM (see section 11.1) and for dynamic knowledge in the DG (see section 11.2).

Regardless of the selected presentation formalism, models created in different phases can be produced from results of several different KA techniques [Parpola, 1999a], as well as iteratively developed during the KB construction process.

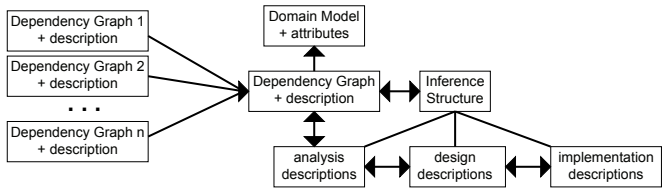


Figure 12.1. The chain of models created in order to build a KB. Seamless transformations are illustrated with arrows. Descriptions of IS are connected to it with plain lines. [Parpola, 1999a].

12.3 Integration of models through transformation

Seamless transformations [Jacobson et al., 1992, Parpola, 1998] are defined between most sequential models in an iterative chain. In SeSKA, automatic and semi-automatic transformations have been defined, usually through combination and other rules, or through the sharing of structures. Different models in the chain (figure 12.1) are described below.

- The *domain model* (DM) contains domain or abstract *concepts* and *relations*. Concepts are described according to a number of *attributes*. The contents of the DM are selected according to what is needed in the DG.

- Initial *dependency graphs* (DG) are acquired from different sources. DGs present *inferential dependencies* between *attributes* of DM concepts. Descriptions can be attached to dependencies.

- The actual DG is a combination of initial DGs.

- *Inference structure* (IS) presents the structure of possible inference sequences performed. The IS is shared among three sets of descriptions.

- Collections of *analysis, design and implementation descriptions* are attached to inferences in the IS. The result is called the *inference model* (IM).

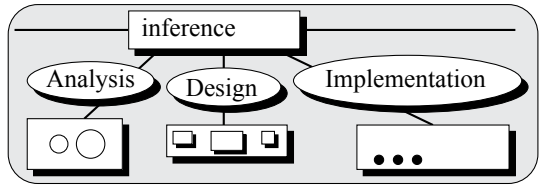


Figure 12.2. Different descriptions (rectangles below ovals) of an individual inference can contain different numbers of blocks.

12.4 Maintenance of the knowledge base structure through a shared skeleton

Blocks of different granularity can be attached to a common IS description (figure 12.2). *Analysis* description blocks consist of major abstract logical components of a description of an inference. *Design* description blocks are the components of a formal description of an inference. In *implementation* description blocks are composed of executable rules or functions.

The collections of all different descriptions, in combination with the IS, form the analysis, design, and implementation models. IM is in effect an integration of three models sharing the IS, which enhances traceability [Parpola, 1998] through inherent *seamless transformations* [Jacobson et al., 1992].

12.5 The possibility of performing inferences in the model

The SeSKA methodology proposes the idea that is implemented in the tool SOOKAT, described in chapter 14.

12.6 Reasons for using metaobject protocols

SeSKA uses metaobject protocols (see chapter 6) for two purposes, i.e., for domain and inference models, in order to be able to (see section 2.4)

- represent and modify application instances and their attributes,
- at run time, create and modify concepts with instances, so that the modifications transfer to instances, and
- use instantiations of the inference structure to apply abstract rules

defined in the inference model to application instances of concept attributes.

Instantiation models are created in order to complete the use of metaobject protocols. Taking full advantage of using the metaobject protocols means being able to simultaneously modify and use the knowledge base.

12.7 Possible implementations of SeSKA

To implement SeSKA, an implementation paradigm should be able to define and modify entities with properties, relations between entities, inheritance, and instances. These facilities enable presentation of metalevel constructions.

At least the OO approach, and conceptual graphs (CG) are acceptable formalisms.

13. The knowledge base construction process

13.1 Overview of the models and their development

The phases of SeSKA are illustrated in figure 12.1. Figure 13.1 gives an overview of the integrated models, using OMT object model notation [Rumbaugh et al., 1991]. Thickened lines distinguish the elements forming the skeleton of the models. Associated descriptions, necessary for the KA task, are presented in rectangles with a thin border.

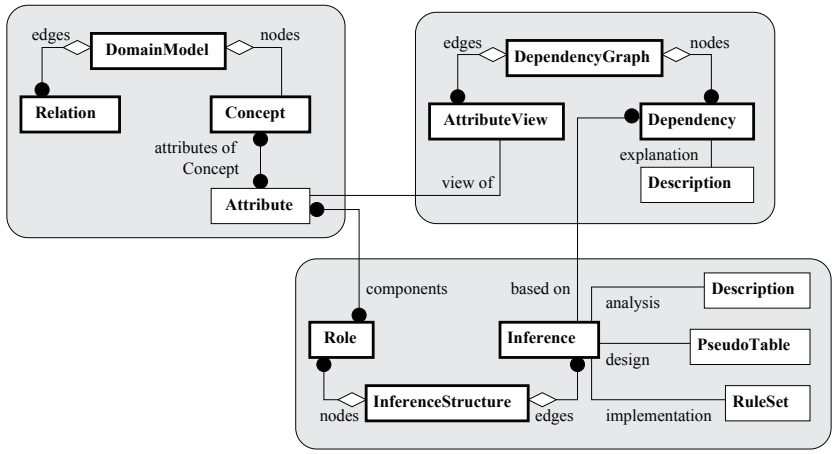


Figure 13.1. Analysis, design and implementation models of a KB.

13.2 Forming initial models describing the domain

The initial *dependency graphs* and the *domain model* are formed based on default value suggestions for and dependency suggestions between concept attributes.

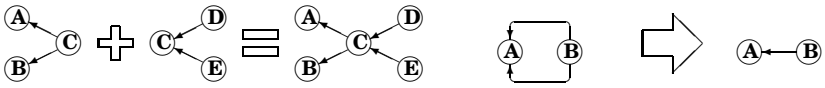


Figure 13.2. Combination rules for dependency graphs, applicable when constructing a KB: *Join*. Two dependency graphs that contain the same node can be joined (left). *Simplify*. One of two duplicate dependencies may be removed, if the same nodes are connected with the same kind of dependencies in both (right). Text descriptions of both of the duplicate dependencies have to be preserved, however.

13.2.1 Combining knowledge in dependency graphs from multiple sources

Knowledge for the initial dependency graphs for an KB is acquired from several knowledge sources that may give differing values. A dependency graph can be created separately for each source. dependency graphs can be seen as a form of conceptual graphs (CG), described by Sowa [Sowa, 1984], considering dependencies as conceptual relations with two arcs.

Complementary dependency graphs can be processed using joining and simplification rules (figure 13.2) [Parpola, 1998].

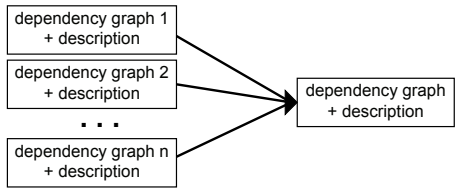


Figure 13.3. Combining acquired dependency graphss in the chain of models to build a KB (figure 12.1).

Complementary dependency graphs can be developed and combined (figure 13.3) using joining and simplification rules [Parpola, 1998], illustrated in figure 13.2 and described below:

Join If two dependency graphs contain identical concepts, then one of the identical concepts can be deleted and all dependencies that had been linked to it, can be linked to the remaining one (both to classes that depend on c, and that c depends on). This can be proved through compatibility of the dependency graphs [Sowa, 1984], definition 3.5.6. Concepts can be also generalized [Sowa, 1984], p. 100. Validity of combining graphs that have not been compatible before generalizations of concepts, depends on the domain, i.e., whether essential properties of the original concept hold also in the generalized con-

cept.

Simplify If two dependencies in a dependency graph are duplicates (of the same type, and defined between the same classes, in the same direction), then one of them may be removed from the dependency graph. Any information, associated with the removed dependency, must be combined with the information associated with the remaining.

As a dependency relation only states the existence of some kind of inferential dependency between two classes, duplicate relations represent redundant information. The associated text information, however, is not (necessarily) redundant.

These rules allow bringing together different fragments of knowledge, even before building a KB, and showing how they might be combined. There will almost certainly be overlapping information left, not removed by the rules. This should be analysed and edited to maintain the KB manageable.

Applying especially the rule 'Join' requires investigations concerning compatibility of the original dependencies, and the context of validity of both original and combined dependencies should be found out. Combination rules may accelerate construction of a KB.

To cope with contradicting or multiple attribute values, SeSKA defines combination heuristics [Parpola, 1999a]. There are no absolute rules for these situations, but common sense can be used. If different experts give different values for the same attribute, e.g. grain size, of a certain concept, e.g. a rock type, the following alternatives should be checked:

- Do some of the experts say they are not sure?
- Does the majority agree? Could someone with a different opinion be wrong?
- Do opinions split into two? Could it be a question of a borderline?
- Could different samples of the same rock type vary in this respect?

Based on the answers, either a single or multiple/alternative values cam be given to the attribute in question.

Again based on common sense, the following heuristics have been defined:

Agreement heuristic If all sources suggest the same value x, set it to be the value.

Majority heuristic If two different values x and y are suggested by n and m sources, and $n > 2 \times m$, n is considered to be the correct value.

Similarity heuristic Compositionally similar concepts have identical values for compositional attributes.

In addition to the joining and simplification rules, rules for removing unnecessary concepts can be used to manipulate dependency graphs. However, they cannot be applied when constructing a KB, since they delete information on a graph. However, concepts can be made invisible in some view without removing them from the underlying graph using these rules:

Remove A concept class c is unnecessary and can be removed (from a view), if either of the following is true:

- Only one concept class b depends on the concept class c, which depends on one or more concept classes d, making c an *intermediate* concept class, and the dependency relation $\star$ is a transitive relation, i.e. if $b \star c$ and $c \star d$ then $b \star d$. Then c can be removed and class b set to depend directly on d and other concept classes that c possibly has depended on.

- No concept class depends on c and it is not among the result concepts, the values of which are requested.

Using the rules for removing unnecessary concepts in views is described in section 14.5.2 and illustrated in figure 14.3.

13.2.2 Forming a domain model from the dependency graph

A domain model is in SeSKA constructed in parallel with the dependency graph(s). Domain concept attributes of the domain model are concepts in the dependency graph, so relevant concepts and attributes are automatically selected to the domain model (figure 13.4). Developing a domain

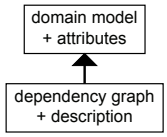


Figure 13.4. Forming a domain model based on dependency graphs in the chain of models to build a KB (figure 12.1).

model in parallel to the dependency graphs means that changes in the domain model are a reflection on any changes in the initial dependency graphs. They can be, and probably are, developed iteratively by adding concepts and attributes and changing attribute value suggestions in the information used to form them.

13.3 Forming an inference model based on dependencies

13.3.1 Forming an initial inference model

An initial *inference model* (IM), built around an *inference structure* (IS), is formed based on dependencies between concept attributes (figure 13.5). This is done by forming a role named <*concept*>-<*attribute*>-*factors* automatically of concept attributes that the attribute <*attribute*> (possibly among others) of the concept <*concept*> depends on. Inferences between roles, and descriptions associated to them, can be created based on dependencies between attributes. This way an IS with analysis descriptions can be created automatically.

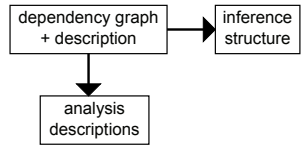


Figure 13.5. Forming the inference structure and the analysis descriptions of inferences based on the dependency graph in the chain of models to build a KB (figure 12.1).

A role in the inference refers to a collection of attributes of a number of domain concepts. These attributes are called member attributes of the role. In addition to automation, also other ways to form an IS can be used, including manual construction and editing. If explanations of dependencies have been given, the combination of explanations of relevant

Premise			Conclusion			
			E	F	G	H
$\wedge$	$\vee$	A + B $<$ C	= A	= red	$>$ 3	= A + B - C
		A $>$ C				
		A + B $<$ D				
$\vee$	...					

Table 13.1. Presentation of a rule table of an inference between roles with member attributes (A, B, C and D) and (E, F, G and H).

dependencies form the first approximation of the analysis description of an inference. An example of forming roles from dependencies is illustrated in figure 13.6.

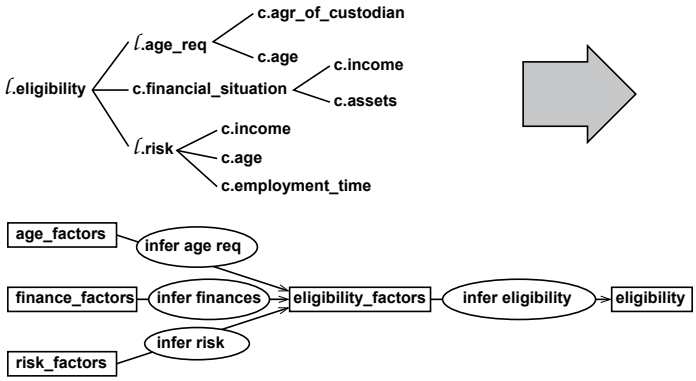


Figure 13.6. Forming roles based on dependencies .

13.3.2 Formalising descriptions of inferences

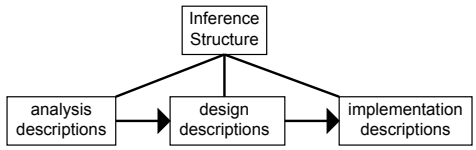


Figure 13.7. Formalizing descriptions of inferences in the chain of models to build a KB (figure 12.1).

Analysis descriptions are formalised to implementation descriptions, e.g. rules, via design descriptions. The process of refinement is iterative and modular. Instances of member attributes of roles become variables in the inference process.

When an inference uses rules, the format of pseudo representation avail-

able is a *rule table*, such as presented in table 13.1, for each rule containing a premise part (on the left) and a conclusion part (on the right). Instances of member attributes of roles are used as variables in the inference process. Different rules are presented one below another. Rules presented in a rule table are directly converted into rules, presented in the implementation formalism used.

Depending on the attribute types, the logical operators on the conclusion part imply things not separately stated in the expressions:

- For well-ordered operand types, operators $<$, $\leq$, $\geq$ and $>$ cause an expression to take the value ordering into account, when the expression is used in a premise. This way it is possible to determine the value of a boolean expression after replacing a constant operand with another value.

- For well-ordered operators formed using conjunction, values of expressions containing their parts in premises can be determined, when used with the same operands. For example $A = B \Rightarrow A \simeq B \Rightarrow A \sim B$.

The above things facilitate making the presentation simple in design descriptions, while they preserve the information. As an example, in the rule table in table 13.1, the conclusion of the rule
$A + B < D \wedge (A + B < C \vee A > C) \Rightarrow G > 3$
automatically makes that rule true also, when it is written
$A + B < D \wedge (A + B < C \vee A > C) \Rightarrow G > 2$.

If functions or procedures are used to define inferences, the input parameters must be member attributes of a premise role. Attributes modified (either through side effects during execution, or as result values) must be member attributes of the conclusion role. For reuse purposes, formal parameters are defined through the ordinal numbers of attributes referenced.

13.3.3 Creating a dependency graph based on an analysis model

An *analysis model* of a KB being constructed consists of the IS and all the analysis descriptions. A dependency graph can be formed from the analysis model. A dependency graph formed this way is not unambiguous, as several different dependency graphs can produce the same analysis model. One way to form a dependency graph is to take the roles con-

nected by an inference, and set all concepts referred to by a conclusion role to depend on all concepts referred by the premise role. The analysis description of the inference can be attached to all dependencies formed, and edited.

13.4 Managing change during development or maintenance

Nature of a KB development is iterative and modular, well suited by seamless transformations.

Often a need for change is acknowledged through implementation errors, or other instability in the design or implementation models. According to principles of seamless development, however, the changes are not made only locally in these models. Rather, the corresponding parts of the analysis model are traced using the shared inference structure. It may frequently be the practice to describe changes that have already been carried out, but it is important to maintain the logical description up to date. However, representing the problem at the abstract (analysis) level may occasionally help in finding the solution.

Different phases of development are visited iteratively, and descriptions corresponding to the phases can be modified on each iteration. Each inference, defined between two or more roles, can proceed according to its individual timetable. According to its stage of development, each inference has one, two or three descriptions attached to it.

A series of OO models, illustrated in figure 12.1 and described in section 14.5, is created and partially modified during the iterative development process.

Part IV

IMPLEMENTATION

14. Implementation of the basic models

14.1 Implementation of the domain model

The domain model is implemented as a network of instances of metaclasses Concept and Relation. Concepts are nodes and Relations edges of the network. Attributes of Concepts are implemented as aggregate attributes referring to class Attribute.

14.2 Implementation of the dependency model

The dependency model is implemented as a network of classes DependencyAttReference and Dependency. Instances of class DependencyAttReference contain both a reference to a domain model Concept and a name of an attribute.

14.3 Implementation of the inference model

The inference model is implemented as a network of instances of metaclasses Role and Inference. Roles are nodes and Inferences edges of the network.

14.4 Implementation of metaobject protocols

14.4.1 Means of implementation

The SOOKAT tool is implemented in the Java programming language, which has only introspection facilities (section 6.1) in a built-in metaob-

ject protocol. That means that the metaobject protocol in it cannot be used to modify the instances in it, unlike e.g. in the Common Lisp metaobject protocol. Due to this, a metaobject protocol with selected reflection facilities is built on top of Java for SOOKAT.

14.4.2 Metaclasses

Metaclasses *Concept*, *Role*, and *Inference* are implemented as normal Java classes in the *domain model* (see section 14.5.1) or the *inference model* (see section 14.5.3).

14.4.3 Classes

Classes in SOOKAT are called application classes in order to distinguish them from Java classes. They are implemented as instances of the metaclasses Concept, Role and Inference.

Instances of the metaclass Concept refer to a group of instances of the class *ConceptAttribute* that define the name, type, and default value of an attribute. The type of a ConceptAttribute denotes

- a unique type name,
- the type of an item in an attribute value, and
- the maximum number of items in a value.

An attribute value is implemented as a list. A type with one as the maximum number of items denotes a *single value*, considered as a special case when referring to the attribute or using it at a user interface. List operations cannot be applied to such an attribute. An attribute value with no items implements an *empty value*. The attribute type is nevertheless known.

The type of a list item in an attribute may be

(a) a primitive type, defining a simple value, or

(b) a type naming an existing Concept subclass, meaning that the value is a reference to an application instance.

Case (b) in a single-value attribute implements a one-to-one relation, whereas with a maximum number of values greater than one, it implements a one-to-many relation. Relations are used in domain and value models. They are implemented as *reference objects*, enabling relations to be either

one or two way relations.

A Role, i.e., an instance of the metaclass Role, refers to a group of ConceptAttributes. An Inference refers to a group of instances of the class *Rule*.

Inheritance hierarchy among application classes is implemented through referring from each application class to its superclass and subclasses. Default values of attributes, as well as dependencies and rules between attributes, are inherited from the nearest superclass but can be overridden. Multiple inheritance is not supported.

14.4.4 Instances

Instances of application classes, called application instances, are implemented as instances of the Java classes *ConceptInstance*, *RoleInstance* and *InferenceInstance* in the *value model* (described in section 14.5.4) and in the *execution model* (described in section 14.5.5). These instances of classes refer to the corresponding instances of the metaclasses Concept, Role, and Inference in the domain model and the inference model.

Instances of the class ConceptInstance define the actual attribute values of application instances. Default values of attributes are expected to be valid, unless they are overridden by referring to new attribute instances. A value of an attribute with a Concept subclass in its type is a reference to a ConceptInstance or a collection of them, thus implementing a relation.

Instances of the classes RoleInstance and InferenceInstance are used to perform actual reasoning among attributes of ConceptInstance instances.

14.5 Architecture of SOOKAT

14.5.1 Domain model

The class *DomainModel* (figure 14.1) contains an inheritance hierarchy of Concept metaclasses in the domain. The class ConceptAttribute can manage both the multiple values acquired from different knowledge sources and the default values worked out. A Concept contains instances of ConceptAttribute. The use of a domain model is described in example 5.

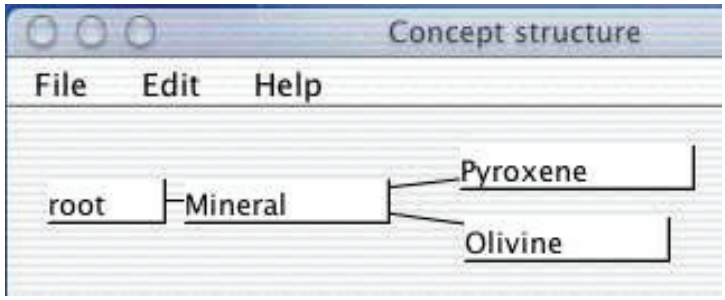


Figure 14.1. A domain model concept graph.

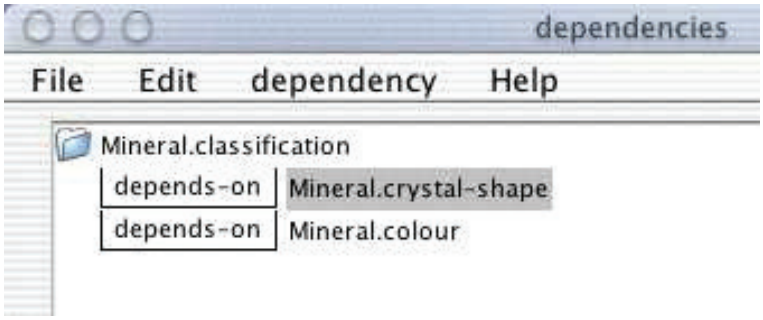


Figure 14.2. A dependency tree presenting a dependency graph. Dependencies defined for a parent concept (Mineral) are inherited to subclasses (e.g. Olivine).

14.5.2 Dependency model

The class *DependencyGraph* (figure 14.2) refers to instances of the classes *AttributeReference* and *Dependency*. An AttributeReference contains a reference to a Concept in the domain model, as well as the name of a ConceptAttribute. A Dependency describes an inferential i.e. logical dependency between attributes of domain concepts. Dependencies of one or several types can be used. The type of a dependency is indicated by referring to an instance of a suitable subclass of the class *DependencyType*. A description is attached to each dependency, as presented in example 3.

Restricted views, showing only selected concepts, can be created for applications. This implements the combination rule remove for dependency

Figure 14.3. The combination rule *Remove* for forming special views of dependency graphs. An unnecessary result concept (left) or an intermediate concept (right) can be removed.

graphs (figure 14.3), without deleting information, as discussed in section 13.2.1.

Example 3. *An instance of the class Dependency, with the default DependencyType, named 'depends-on', is created between Mineral.classification and both Mineral.colour and Mineral.crystal-shape. To the dependency is attached the description:*

"If Mineral.colour is green and Mineral.crystal-shape is octagonal then Mineral.classification is Olivine"

14.5.3 Inference model

The model, managed through the class *InferenceModel*, is based on a network called the *inference structure*, which describes the structure of possible inferences through instances of the metaclasses Role and Inference. Inference defines, for each description level, a separate *aggregate* attribute:

An analysis-level description is an abstract textual description. The initial analysis-level description of an inference is formed as a combination of descriptions of the dependencies the inference is based on. The description may be presented as a table.

A design-level description is a semi-formal presentation of the analysis-level description. In SOOKAT, it is implemented as a *rule table* (table 14.1).

An implementation-level description is composed of abstract descriptions of executable rules, implemented using the class *Rule*, containing

- a *premise expression*, i.e. an instance of the class *BooleanExpression*, defining a logical operator, as well as operands that may be
  - Expressions,
  - RoleAttributeReferences, acting as variables, or
  - arithmetic (integer) or logical (boolean) constants and references to variables evaluating to primitive types,

Premise		Conclusion
	Mineral.colour = green	Mineral.classification = Olivine
∧	Mineral.crystal-shape = octagonal	

Table 14.1. A rule table for inferring Mineral.classification, formed by formalising descriptions of dependencies appropriately. The dependency between Mineral.classification and both Mineral.colour and Mineral.crystal-shape is formalised. The logical connector ∧ (and) is presented before the premises, connecting them, and the operators used in premises are written between the attribute names and values. Conclusion attribute(s) with value(s) is/are presented on separate rows.

- a *reference* to the *conclusion attribute*, i.e. a RoleAttributeReference instance, and

- a *formula* for obtaining the *conclusion value* that is an instance of the class *ValueExpression* which is a subclass of the class *ArithmeticExpression*. The conclusion attribute reference and the conclusion value form a BooleanExpression with an = (equals) operator.

Example 4. *The instance of the class Inference in example 6. has the descriptions:*

- *analysis level: table containing the statement of reasoning Mineral.classification.*

- *design level: rule-table presentation of the analysis-level description (see table 14.1)*

- *implementation level: an instance of the class Rule with the components:*

  premise expression: *BooleanExpression instance, containing the logical operator ∧, and two operands that are themselves instances of BooleanExpression. The operator in both is = (equals, also ≃ could be used with observed colours) and the operands are:*

  - *in the first one, the RoleAttributeReference instance 'Mineral.colour' and a constant representing the colour 'green', and*
  - *in the second, the RoleAttributeReference 'Mineral.crystal-shape' and a constant representing the shape 'octagonal'.*

  reference to conclusion attribute: *RoleAttributeReference instance 'Min-*

eral.classification'.

formula for obtaining the conclusion value: *constant reference to the ConceptInstance Olivine.*

14.5.4 Value model

In the value model, application instances are represented by instances of the Java class ConceptInstance. The values of their attributes are the possibly variable given values (see example 5.) and the different possible conclusion values from the inferences. The latter depend on the execution model, in addition to the inference model.

14.5.5 Execution model

The execution model contains instances of the Java classes RoleInstance and InferenceInstance. Messages are sent between these application instances in an order controlled by an instance of some subclass of the class ControlObject (see section 18.2).

InferenceInstance instances adjust abstract rules. RoleAttributeReferences are replaced with corresponding values or references to attributes of application instances in the value model.

The use of an execution model is described in example 6.

14.6 Co-operative building, adaptation, and evolution of abstract models of a KB

14.6.1 Tool support for acquiring the domain model

Suggestions for attribute values of concepts can be inserted in an arbitrary order by different knowledge sources. All suggestions are stored in instances of the class ConceptAttribute, a subclass of the class Attribute.

14.6.2 Tool support for acquiring the dependency graph

To eliminate heterogeneity in AttributeReference names, Concepts can be selected from among the ones in the domain model, as well as an attribute

name from among those in a selected Concept. DependencyTypes can be selected from among the instances of it or its subclasses.

The combination of dependency graphs acquired from different sources is implicit in SOOKAT. Combination rules (figure 13.2) [Parpola, 1998] can be used incrementally, even when several knowledge sources are considered in parallel, assuming the context to be the same [Parpola, 2002, Parpola, 2001]. Dependencies are joined automatically. Simplification is performed semi-automatically when SOOKAT collects dependencies. SOOKAT simply joins descriptions of varying dependencies and the user can remove duplicate descriptions. The descriptions and sources of the original dependencies are maintained. Descriptions of dependencies can also be augmented with lists of suitable contexts.

14.6.3 Semi-automatic formation of an initial inference model based on dependencies

The dependency graph, with its descriptions, is used in forming the initial inference model, called the *analysis model*, consisting of the inference structure and abstract descriptions taken directly from dependencies (see section 13.3). Inference structure formation is triggered from the user interface.

14.6.4 Content management

The informal, semi-informal, and formal descriptions are stored as attributes of instances of the class Inference, giving abstract descriptions of inferences in the inference model. Thus, formal and informal descriptions can be stored contiguously, as different descriptions can simultaneously be at different stages of development. Transformations between different descriptions of an inference are performed semi-automatically.

Different models can be modified in the user interface of SOOKAT, and models can be saved in a format that can be exported. Changes made can to some extent be propagated to other models [Parpola, 1999a].

14.7 Storing and transferring models

A domain or inference model, or an application (execution or value model), or a part of either of them can be stored for later use, or be transferred to another application. With respect to changing and storing, a model can be

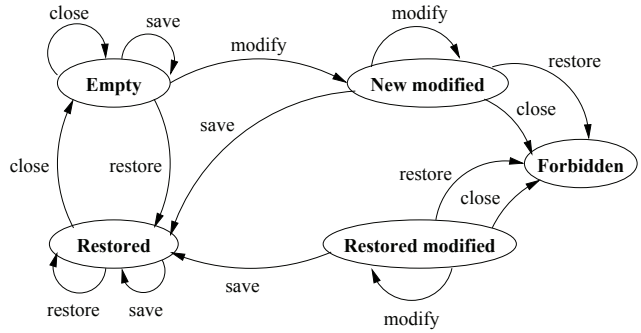


Figure 14.4. The state diagram for using file or other storage in a persistent SOOKAT model.

modified, restored, saved or closed. At any state, as many as possible of these operations are available to a user. The state diagram is presented in figure 14.4. A similar approach is used separately for individual objects in a model.

Specifically, the SOOKAT implementation uses Java object serialisation for storing the models. For performance, a database could be used instead of a file system. To utilize models outside SOOKAT, a standard way of storing metaobjects should be selected, such as presented in section 6.2.2.

When using serialisation, each complex main object in any model is stored separately, to support modularity and reusability. Due to that, all references to other complex objects are referred to with unique identifiers, that is, names, upon serialisation. Therefore the names of the objects in values of attributes have to be unique. This is quaranteed when building an application by checking object names upon creation. Uniqueness of object names between applications is ensured by using application names as a part of unique identifiers of objects in attribute values upon serialising an object containing them. After deserialisation, identifiers are converted to valid object references again.

15. Implementation of the examples

15.1 Mineral classification toy application

Example 5. *Instances of the metaclass Concept are used to present the application class* Mineral*, as well as its subclasses* Olivine *and* Pyroxene*. The three Concept instances know about their subclasses and superclass.*

The application class Mineral defines the attributes colour*,* crystal-shape *and* classification *which all have empty default values. Olivine overrides the attributes colour and crystal-shape with new attributes having the default values 'green' and 'octagonal'.*

Instances of the application class Mineral and its subclasses, which are all instances of metaclass Concept, are implemented in the value model as instances of the class ConceptInstance. Instance mineral#23 *of Mineral overrides the attributes colour and crystal-shape with new attributes having the values 'green' and 'octagonal', corresponding to its appearance.*

15.1.1 Forming initial dependencies

Example 6. *Two instances of the metaclass Role are defined. The first refers to the attributes colour and crystal-shape of Mineral and the second to the attribute classification of Mineral.*

These two application classes of Role are called classification-factor-role *and* classification-role *and contain premise and conclusion attributes of the abstract inference referring to the abstract rule*

"If Mineral.colour is green and Mineral.crystal-shape is octagonal
then Mineral. classification is Olivine."
Instances of the two application classes contain premise and conclusion

attributes of actual inferences performed, i.e. references to the attributes colour and crystal-shape, as classification, of mineral#23.

An instance of the metaclass Inference is defined between classification-factor-role and classification-role. It gives an abstract rule as to how the value of the attribute classification of Mineral can be reasoned on the basis of the attributes colour and crystal-shape of Mineral.

Instances of InferenceInstance transform the references to abstract premise and conclusion attributes into references to the actual premise and conclusion attributes used. The abstract rule becomes

"If mineral#23.colour is green and mineral#23.crystal-shape is octagonal then mineral#23.classification is Olivine"

15.2 Implementation of Sisyphus III rock classification

Instances of metaclass Concept are used to present the application classes *Rock* and *Mineral* as well as their subclasses representing different rock and mineral categories. Concept instances know about their subclasses and superclass.

Application classes Rock and Mineral define attributes with empty default values for their measurable properties and other properties of interest. Subclasses of Rock and Mineral (representing different rock and mineral categories) override the attributes for their measurable properties with new attributes having default values typical of each rock or mineral category.

Mineral and its subclasses defined in the mineral classification toy application (section 15.1) can be utilised in the Sisyphus III example by

- importing the Mineral model,
- adding an attribute to the Mineral, such as *Mineral.percentage* denoting the amount of it in a Rock, and
- making subclasses of Mineral, such as *Olivine*, *Quartz* and *Silica*, types of aggregate attributes such as Rock.olivines, Rock.quartz and Rock.silica, respectively.

rock.olivines	→	rock.shade
rock.quartz	→	rock.silica
rock.silica	→	rock.acidity
rock.shade	→	rock.acidity
rock.shade	→	rock.quartz

Table 15.1. Dependency graphs achieved in the acquisition process, from different sources, are combined according to the combination rules (presented in figure 13.2.)

This model makes properties of minerals available in the Sisyphus III example, adding the necessary information about the amount of a Mineral in a Rock. It is also an example of reusing models between applications.

15.2.1 Forming initial dependencies

The feature values of a rock or mineral sample are presented as values of attributes of an instance of either application class (Rock or Mineral). These values are encoded into a dependency graph in which names of dependencies are the names of feature values and values attached to these dependencies values of the features.

Dependencies are formed in either of the following ways:

Recognition based on feature values All experts, or majority of the experts, suggest the same values.

Recognition based on rules Attributes can be given values. Some feature values have to be reasoned based on values of other features, by using rules, explained in the interviews. These rules may have to be applied several times.

Example of using combination rules for Rocks

Example 7. *Combination of dependency graphs, presenting dependencies between colour, minerals, silica content and acidity of Rocks.*

When the dependency graphs, presented in table 15.1, that are results of KA (from multiple sources), are joined, the dependency graph in figure 15.1 is achieved. Dependency graphs containing values of all or some attributes can be used, for instance, in testing the KB before constructing it, or in forcing some attribute values to divide inference between major parts of the KB.

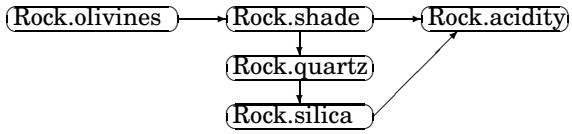


Figure 15.1. Combination of simple dependency graphs.

15.2.2 Forming the domain model simultaneously with the dependency graphs

Example 8. *Instances of metaclass Concept are used to present the application classes* Rock *and* Mineral *as well as their subclasses. The Concept instances know about their subclasses and superclass.*

Application classes Rock and Mineral define attributes with empty default values for their measurable properties. Subclasses of these application classes override the attributes with new attributes having default values matching general feature values typical of different rock and mineral types. Instances of application classes Rock and Mineral and their subclasses, that are all instances of metaclass Concept, are used in inferences.

Instances of rocks and minerals are implemented in the value model as instances of class ConceptInstance. Instances rock#11 *and* mineral#22 *of ConceptInstance override the attributes for their measurable properties with new attributes having values corresponding to their actual properties.*

15.3 Implementing dietary management of MS

15.3.1 Implementation of the domain model for the dietary management of MS

Knowledge concerning dietary management of *multiple sclerosis* (MS) is presented using two instances of metaclass Concept, i.e. two application classes (that may have subclasses):

Body denoting the physical body of a person with MS, and

Diet denoting dietary intake of certain ingredients of interest.

Parts of concepts can be described through aggregate attributes, as described in chapter 14.

Concept instances know about their subclasses and superclass. Application classes Body and Diet define attributes, needed in classes of an in-

tended application. These attributes are defined using Attribute classes with empty default values.

15.3.2 Forming initial dependencies

Some research results, concerning certain groups of fatty acids, vitamins and minerals, have below been presented in narrow-focused dependency graphs (figures 15.2 – 15.9), presenting attributes of concepts in the form <Concept>.<attribute>. Explanatory text fragments, associated with dependencies, are presented in captions. A few generalizations, e.g., replacing linoleic acid with Ω6, have been made, based on basic properties of the Ω6 group [FAO and WHO, 1994]. Captions show the original form. All figures describe the situation in a person with MS, possibly compared with a person without MS.

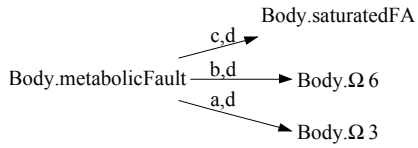


Figure 15.2. A metabolic absorption fault causes the levels of (a) Ω3 < normal [Cunnane et al., 1989, Neu, 1985], (b) Ω6 < normal [Cunnane et al., 1989, Fisher et al., 1987, Navarro and Segura, 1988, Navarro and Segura, 1989], and (c) saturated FA > normal [Navarro and Segura, 1988, Navarro and Segura, 1989]. (d) A predisposing factor causing MS seems to be related to a disturbance of the lipid and fatty acid metabolism [Neu, 1985], A common aspect appears to be a lipid imbalance involving the essential fatty acids (EFA), linoleic and linolenic, and trace fatty acids which result from faulty lipid metabolism [Marshall, 1991].

15.3.3 Forming the domain model simultaneously with the dependency graphs

The domain model, illustrated in figure 15.10 has been formed, using the concepts and attributes appearing in the dependency graphs of the previous section. Simultaneous formation of dependency graphs reveals what attributes the concepts in the domain model should have.

Both people with MS and their individual diets are implemented as instances of metaclass Concept, Different patients and diets are defined as instances of subclasses of the abstract classes Body and Diet.

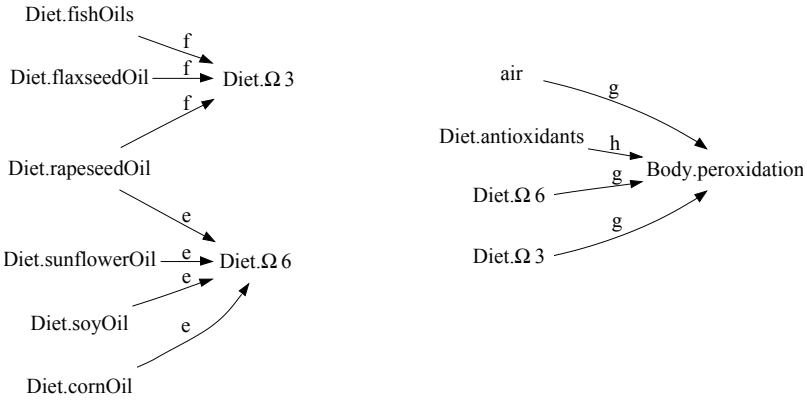


Figure 15.3. (e) Some vegetable seed oils, such as sunflower oil, corn oil, soy oil and rapeseed oil are main dietary sources of Ω6 fatty acids, that are based on linoleic acid [Hyvönen et al., 1993]. (f) Flaxseed oil and rape-seed oil are main dietary sources of linolenic acid [USDA Nutrient Data Laboratory, 2004, Hyvönen et al., 1993], an Ω3 fatty acid. Other Ω3 fatty acids, i.e., eicosapentaenoic (EPA) and docosahexaenoic (DHA), are also available in fish oils [KTL Nutrition Unit, 2004]. (g) EFA are easily peroxidized in air. (h) Vitamin E prevents peroxidation [Sinclair, 1984].

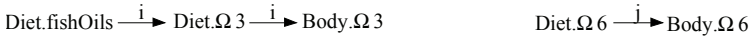


Figure 15.4. (i) Ω3 is absorbed from dietary fish oils [Nightingale et al., 1990]. (j) Linoleic acid showed significant correlations with diet [Fitzgerald et al., 1987].

Figure 15.5. (k) Supplementation of Ω3 and Ω6 fatty acids caused reduction of the severity and frequency of relapses and a mild overall benefit [Bates, 1990].

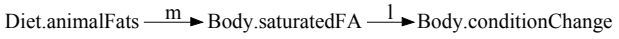


Figure 15.6. (l) Use of saturated FA increased deterioration and lethality. (m) Animal fats are saturated [Swank, 1991].

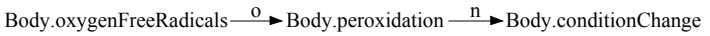


Figure 15.7. (n) Pentane (a peroxidation product) raised exactly during relapses (exacerbations). (o) It has been concluded that oxygen free radical activity is enhanced during exacerbation of multiple sclerosis [Toshniwal and Zarling, 1992].

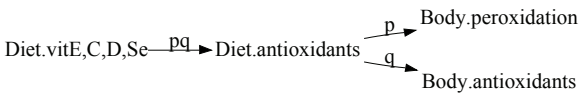


Figure 15.8. Antioxidants (selenium, vitamin E and vitamin C) normalized abnormalities, i.e., (p) lowered increased peroxidation rates and (q) raised lowered selenium levels [Clausen et al., 1988].

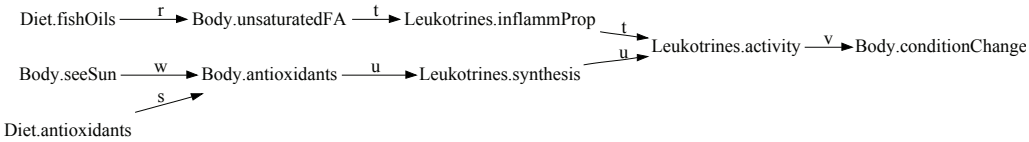


Figure 15.9. (r) Dietary fish oils may be beneficial. (s) Dietary antioxidants may be beneficial. (t) Fish oils lead to production of leukotrines with less inflammatory properties. (u) Antioxidants inhibit leukotrine synthesis. (v) Leukotrines might be the underlying cause of certain symptoms (retrobulbar neuritis). (w) Visual solar radiation releases rhodopsin with vitamin A, in the visual pigment of retina [Hutter, 1993].

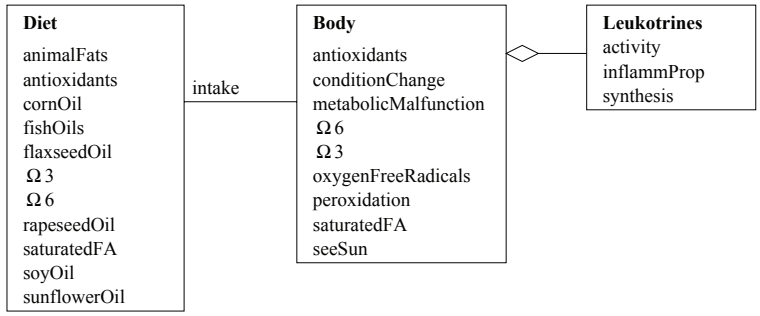


Figure 15.10. OMT presentation [Rumbaugh et al., 1991] of the domain model for nutrition of a person with MS.

15.3.4 Combining dependency graphs from different sources

An example of combining dependency graphs will now be given. Dependency graphs in figures 15.2 and 15.4, are combined, based on the appearance of node 'Body.Ω6 in both dependency graphs, and rule 'join' in section 13.2. The resulting dependency graph is illustrated in figure 15.11.

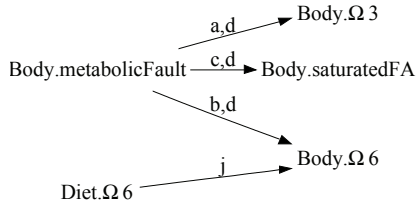


Figure 15.11. Combination of dependency graphs in figures 15.2 and 15.4 (j).

After joining dependency graphs in figures 15.4 (i) and 15.3, there are duplicate relations between 'Diet.fishOils' and 'Diet.Ω3', so rule 'simplify' has to be used. After joining also the dependency graphs in figures 15.5 and 15.6, the dependency graph in figure 15.12 is achieved.

Starting from a different point, dependency graphs in figures 15.3, 15.7, 15.8, and 15.9 are combined. After joining dependency graphs in figures 15.7 and 15.8 by node 'Body.peroxidation', the dependency graph in

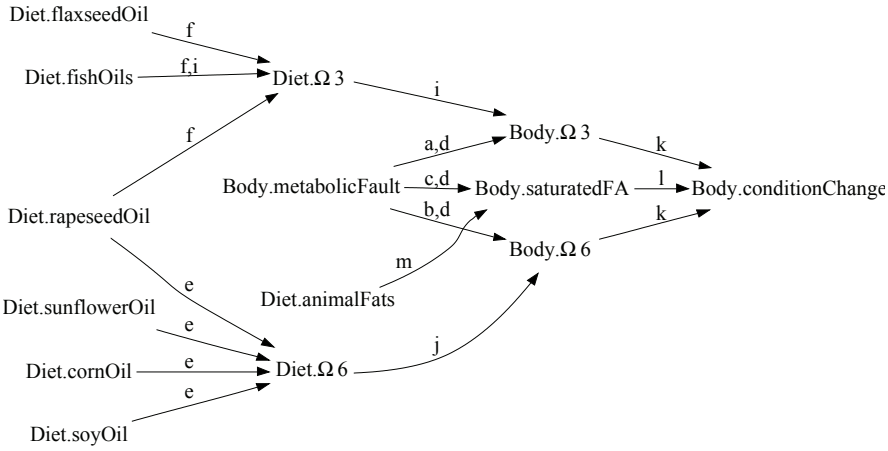


Figure 15.12. Combined dependency graph after adding the dependency graphs in figures 15.3, 15.4 (i), 15.5, and 15.6.

figure 15.9 can also be joined. There are several nodes that have to be joined, 'Condition.degradation', 'Diet.antioxidants' and 'Body.antioxidants', causing a duplicate relation to be simplified between 'Diet.antioxidants' and 'Body.antioxidants' The combined dependency graph is illustrated in figure 15.13.

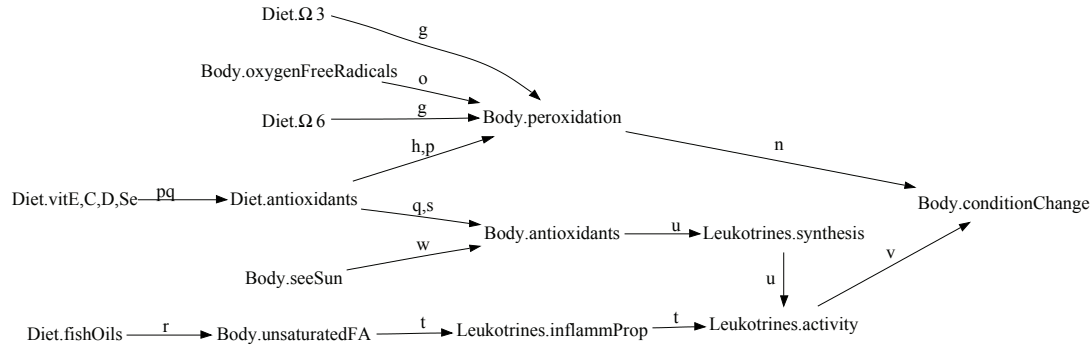


Figure 15.13. Combination of the dependency graphs in figures 15.3, 15.7, 15.8 and 15.9.

When graphs presented in figures 15.2 – 15.9 are combined, the dependency graph illustrated in figure 15.14 is achieved. Explanations associated with dependencies can be found from captions of dependency graph figures in section 15.3.2.

15.3.5 Forming the initial knowledge base

The IS, illustrated in figure 15.15, is based mainly on dependencies of figure 15.14, using the technique described in section 13.3: combining

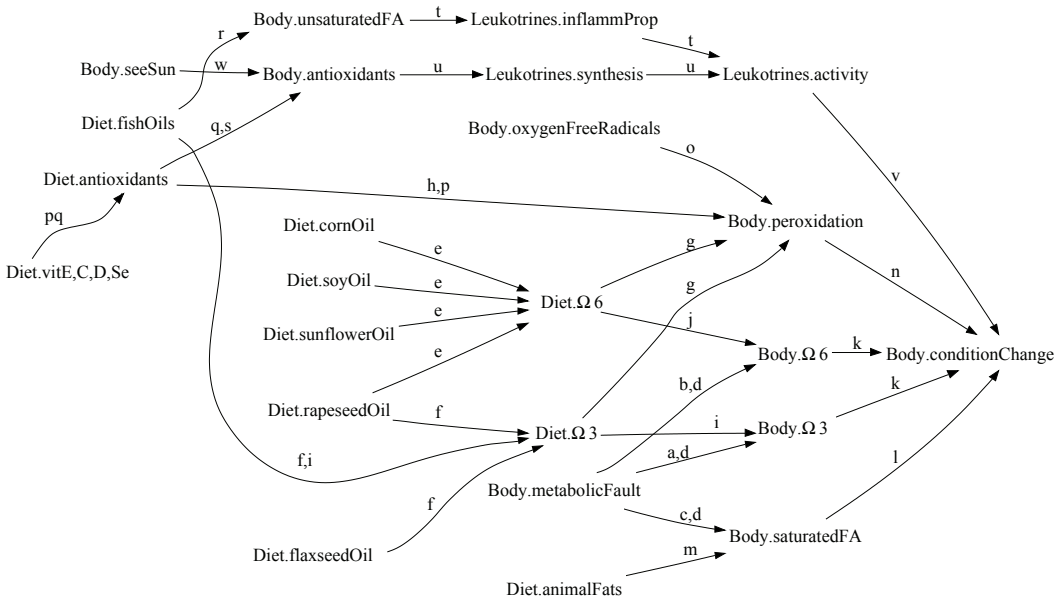


Figure 15.14. The combined dependency graph.The letters associated to dependency arcs appear in dependency graphs presented onwards from section 15.3.3.

attributes, that a certain attribute depends on, as a role. Roles have then been given identifiers and descriptive names. Contents of roles is described in the caption. Inferences simply refer to the dependencies between components of roles. Explanations of the inferences are those of component dependencies.

15.3.6 Formalizing descriptions

Only an example is taken of rule formalisation. The verbal description of rule described by dependencies b, d and j (figure 15.11) is "(Due to a fault in FA metabolism) Body.Ω6 <NORMAL; Ω6 is absorbed from dietary supplementation." The same thing can be presented in a design table 15.2 (a), which directly converts into the rules in table 15.2 (b), presented in the implementation language. As mentioned in section 13.3.2, the development process is iterative and modular.

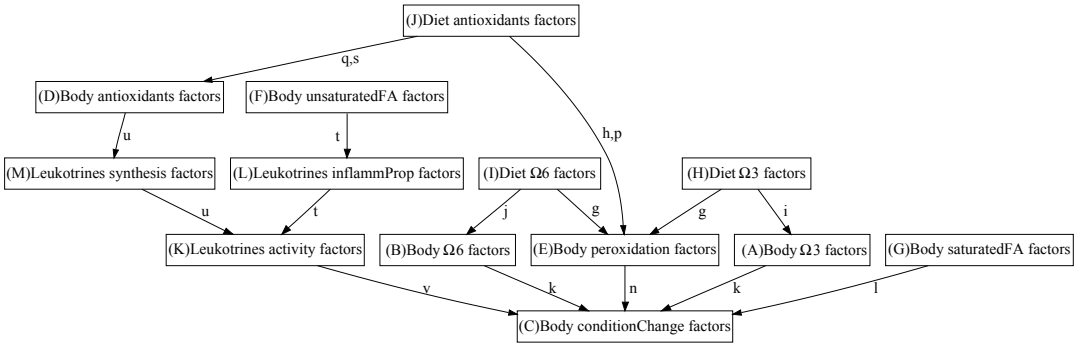


Figure 15.15. The initial inference structure with analysis descriptions. The letters associated with dependency arcs refer to initial dependency graphs. Roles: (A) {Diet.Ω3, Body.metabolicFault}; (B) {Diet.Ω6, Body.metabolicFault}; (C) {Leukotrines.activity, Body.peroxidation, Body.saturatedFA, Body.Ω3, Body.Ω6}; (D) {Body.seeSun, Diet.antioxidants}; (E) {Diet.Ω6, Diet.Ω3, Body.oxygenFreeRadicals, Diet.antioxidants}; (F) {Diet.fishOils}; (G) {Diet.animalFats, Body.metabolicFault}; (H) {Diet.fishOils, Diet.rapeseedOil, Diet.flaxseedOil}; (I) {Diet.sunflowerOil, Diet.cornOil, Diet.soyOil, Diet.rapeseedOil}; (J) {Diet.vitE,C,D,Se}; (K) {Leukotrines.inflammProp, Leukotrines.synthesis}; (L) {Body.unsaturatedFA}; (M) {Body.antioxidants}.

		Body.Ω6
∧	Body.metabolicFault = true	is lowered
	Diet.Ω6 = normal	
∧	Body.metabolicFault = true	is normal
	Diet.Ω6 is supplemented	

if Body.metabolicFault = true
and Diet.Ω6 = normal
then Body.Ω6 is lowered

if Body.metabolicFault = true
and Diet.Ω6 is supplemented
then Body.Ω6 is normal

Table 15.2. Combining acquired dependency graphs in SeSKA. (a) Rule table describing rule b,d,j, that is, transformation of role I to B (on the left). (b) Rules formed from the rule table beside (on the right).

Part V

INSTANTIATION

16. Instantiation of models

In order to be able to perform inferences in an application, the domain model (described in section 14.5.1) and inference model (described in section 14.5.3) have to be instantiated to form the value model and execution model (described in sections 14.5.4 and 14.5.5).

16.1 Instantiating the domain model to introduce the value model

The Concepts in the domain model are gone through by the tool in order to remind the user of the items to be instantiated. For the desired concepts, the user can create one or more application instances with specific names and attribute values.

16.2 Instantiating the inference model to introduce the execution model

Roles in the inference model will be instantiated with selected collections of attribute instances. InferenceInstance instances between RoleInstance instances trigger rules of Inference metaobjects, applied to the ConceptInstance instance attributes, indicated by the RoleInstance instances.

A reason for defining Inference as a metaobject is that it provides a convenient way of collecting together a group of InferenceInstances using the same rules. In this way, modifying a rule in a subclass of Inference affects all of its InferenceInstances at a time.

17. Instantiation of the examples

17.1 Instantiation of mineral classification toy application

Instances of application class Mineral and its subclasses, that are all instances of metaclass Concept, are implemented in the value model as instances of class ConceptInstance. Instance *mineral#23* of Mineral overrides the attributes colour and crystal-shape with new attributes having values 'green' and 'octagonal', corresponding to its appearance.

17.2 Instantiation of Sisyphus III rock classification

Instances of application classes 'Rock' and 'Mineral', that are instances of metaclass 'Concept', are implemented in the value model as instances of metaclass 'ConceptInstance'. They contain attributes for conclusions of inferences, that can get different values in different application subclasses. The values of the conclusion attributes are initially empty (section 14.4.3) and they are used in reasoning as described in section 18.5. Additional attributes are defined in the concept 'Rock' to denote different minerals and their percentages, as described in section 15.2.

Example 9. *The attribute colour of application class 'Rock' gets the value 'black' in instance* rock#1 *where value of the attribute 'rocktype' is 'Basalt' and the value 'leuocratic' in instance* rock#2 *where value of the attribute 'rocktype' is 'Granite'.*

Instances of 'Rock' and 'Mineral' may override their attributes with new attributes having values, corresponding to their measurable properties or appearance.

Example 10. *Instance* mineral#23 *overrides the attributes 'colour' and 'crystal-shape' with new attributes having values 'green' and 'octagonal', according to its appearance.*

Example 11. *Instance* rock#21 *overrides the attribute 'olivine_percentage' with the actual value '100'. This means that rocktype is 'Dunite'.*

17.3 Instantiating dietary management of MS

Desired numbers of instances can be created of classes BodyInstance and DietInstance, as well as their subclasses, denoting individual people with MS and diets. These classes are subclasses of ConceptInstance.

Empty values are initially generated for attributes

- omega-3 and omega-6 unsaturated fatty acids, i.e., the essential fatty acids,
- vitamin B12,
- vitamin D,
- antioxidants,
- lactic acid, and
- fresh fruit and vegetables

of class DietInstance. These values can later be overridden with new attributes having values corresponding to the actual situation in the diet of a person with MS.

The dependency graph illustrated in figure 15.14 can be instantiated for individual people and individual dietary regimens as described in this section.

Part VI

REASONING

18. Basic principles of reasoning

Basic principles of reasoning are illustrated using examples in the mineral classification toy application.

18.1 The message sending and assignment mechanism

Inferences are performed according to principles that have been adopted from the KADS methodology [Hesketh and Barrett, 1989, Schreiber et al., 1999] and modified. InferenceInstance instances between RoleInstance instances trigger rules of corresponding Inference instances, applied to the desired ConceptInstance attributes.

Assignments are made to values of Attributes of ConceptInstance instances in the value model, considered as variables to which either values of, or references to, the actual application instance attributes are assigned by InferenceInstance instances.

It has been proved that reasoning through the assignment of concept values holds the power of first-order logic [Wetter, 1990]. Replacing concepts with attributes of concepts makes only a small addition to the proof.

Example 12. *A green mineral with octagonal crystals needs to be identified, using a KB containing the roles r1 (Classification-role) and r2 (Classification-factor-role) as well as the inference inf1, all described in example 6.*

When 'inf1' is performed, the concept Mineral in it is taken as a variable, to which the application instance mineral#23 is assigned. Attributes referred to from the premise role 'r2' are replaced with links to mineral#23.colour and mineral#23.crystal-shape, which have the values 'green' and 'octagonal', respectively. The value of the attribute mineral#23.classification is assigned to be 'Olivine'. The execution of the task is illustrated in figure 18.1.

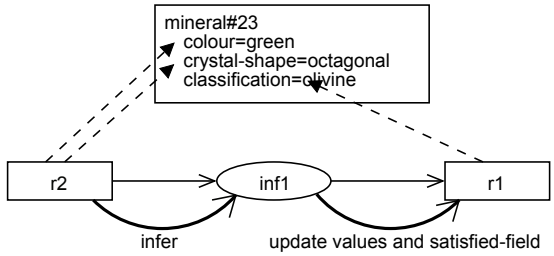


Figure 18.1. Messages sent in inferring the attribute value mineral#23.classification, using forward chaining.

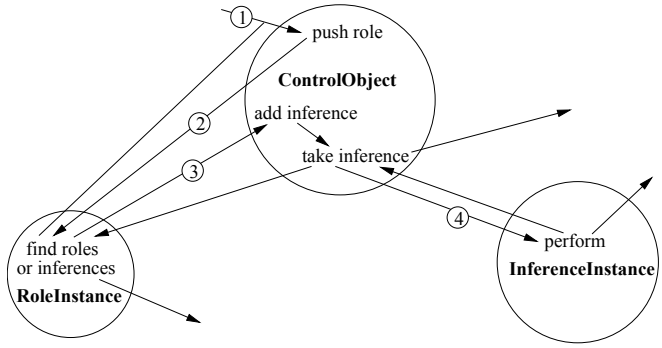


Figure 18.2. Messages sent between the control object, a role instance and an inference instance during backward chaining.

The structure of possible inferences is defined by the instantiated inference structure.

18.2 Control objects

This section is reproduced from 10.1007/s10115-004-0181-6, Knowledge and Information Systems with permission. All rights reserved.

A *ControlObject* (CO), specific to the chosen inference strategy, is used to control message sending between RoleInstance and InferenceInstance instances.

Actual inferencing takes place when COs for different inference strategies also guide the overall process as individual tasks. Message sending among CO, RoleInstance, and InferenceInstance instances, i.e. objects, is controlled by the CO. Suitable actions in RoleInstance and InferenceInstance objects are also triggered by the CO.

The actual order of inferences to be performed is determined by the inference strategy used. The performance of an inference according to backward chaining is illustrated in figure 18.2.

18.2.1 Control object for forward and backward chaining

The main data structures used in controlling inference according to forward or backward chaining are a goal stack, containing RoleInstances, and an inference queue, containing InferenceInstances. In backward chaining, a stack for premise RoleInstances is also used.

In both backward and forward chaining, InferenceInstances with satisfied premises (RoleInstances) are added to the inference queue. A premise is satisfied when the attributes it refers to have values. In backward chaining, the goal RoleInstances and their child RoleInstances are pushed to the goal stack, as long as they are unsatisfied.

Figure 18.2 illustrates messages sent between objects belonging to different classes when controlled by a CO used for backward chaining. Other COs, used for reasoning according to PSMs, are described in section 18.4.

Example 13. *The execution of the mineral classification task in example 12., illustrated in figure 18.2, is described in detail in table 18.1. Messages sent between objects, as well as tasks performed by objects during the execution of the problem-solving task, are specified.*

18.2.2 Generating explanations

In order for it to be possible to produce step-by-step explanations, each inference performed (by executing its implementation description) is provided with a time stamp and saved to a storage with it. Explanations are produced on the basis of

- references from time stamps to InferenceInstances,
- analysis descriptions of the corresponding Inferences, and
- values of Attributes of ConceptInstances referred to from RoleInstances referred to from InferenceInstances.

Example 14. *The explanation produced of inferring the value 'Olivine' of the attribute mineral#23.classification is based on the analysis description of the single inference performed. The following explanation is produced:*

"The value 'Olivine' for the attribute mineral#23.classification was achieved

- CO receives a request to infer ‘r2’ (referring to mineral#23.classification).
- CO adds ‘r2’ to the stack of pending tasks.
- CO demands ‘r2’ to report roles it depends on or inferences it invokes.
- ‘r2’ finds ‘r1’ (referring to mineral#23.colour and mineral#23.crystal-shape) and requests CO to infer it.
- CO adds ‘r1’ to the stack of pending tasks.
- CO demands ‘r1’ to report the roles it depends on or the inferences it invokes.
- ‘r1’ is satisfied (mineral#23.colour is ‘green’ and mineral#23.crystal-shape is ‘octagonal’) and returns ‘inf1’.
- ‘r1’ asks CO to perform ‘inf1’, offering its values as input.
- CO adds ‘inf1’ to the queue of pending inferences.
- CO pops ‘inf1’ from the queue of pending inferences.
- CO invokes ‘inf1’ to perform.
- The rule in ‘inf1’ is executed and value is returned (mineral#23.classification is ‘Olivine’).

Table 18.1. Messages sent in inferring the value of attribute mineral#23.classification during backward chaining.

as the value of the attribute mineral#23.colour is 'green' and the value of the attribute mineral#23.crystal-shape is 'octagonal' and the rule

"If Mineral.colour is green and Mineral.crystal-shape is octagonal then Mineral.classification is Olivine"

was applied."

18.3 Using protocols for reasoning

Here a *protocol* means a documented series of reasoning stages that are gone through when a solution based on some premise values is reached [Ericsson and Simon, 1984]. In other words, a protocol goes through an instantiation of one possible path (a sequence of inference steps, i.e. reasoning stages) through the inference structure. Reasoning stages are called protocol phases.

In SOOKAT, protocols are presented using dependencies. Each protocol phase depends on the attribute(s) reasoned during it, and on the previous phase. In other words, a DependencyGraph (see sections 13.2 & 14.5.2) can describe the ordering of reasoning steps, in addition to logical dependencies between attributes.

Dependencies between the attributes involved are included in the dependency graph with informal descriptions of the reasoning, using the values mentioned in the protocol phases. If a dependency already exists, its analysis description is expanded.

The metaclass *Protocol* is defined as a subclass of the metaclass Concept. The class *ProtocolPhase* is, in turn, defined as a subclass of the class ConceptAttribute. Dependencies (instances of the class Dependency) are added to the dependency graph between attribute references (instances of the class DependencyAttReference) and/or protocol phases.

As Protocol is a metaclass, the following are possible:

1. dynamic creation of Protocol instances,

2. inheritance among Protocol instances, and

3. instantiation of Protocol instances (e.g. MineralClassificationProtocol instances).

The use of the second or third possibilities could increase the flexibility of the system through enabling the association of different and alternative COs to instances of Protocols.

18.3.1 Alternative instantiations of attributes used in protocols

Protocol phases are extensible through alternative values for the attributes involved in dependencies, which have arisen in different connections. Suitable instantiations of concept attributes, referred to from a dependency graph, can be used to enlarge the scope of a protocol phase and possibly a protocol.

In other words, protocols can be expanded through alternative application instances of the protocols, i.e. by referring to alternative result instances of the metaclass Concept and alternative starting values, but retaining the attribute names.

Example 15. *The mineral classification example can be extended by adding the information for classifying the mineral* Pyroxene*: Mineral.colour is 'green' and Mineral.crystal-shape is 'tabular'.*

18.3.2 Inference according to protocols

The premise and conclusion attributes involved in each phase are essential in a protocol applied to a reasoning task. The values of these attributes are taken from an instantiation of the path in the dependency graph. The forward and backward chaining strategies can also be used when reasoning according to protocols.

The reachability of different solutions and coverage of existing material (dependencies) have to be checked. In other words, a path must exist in the dependency graph for reachiing each solution in the application domain.

Example 16. *The possible solutions in the mineral classification example domain are 'Olivine' and 'Pyroxene'. As dependencies exist for reaching both, coverage of the material is sufficient.*

18.4 Inference using PSMs

This section is reproduced from 10.1007/s10115-004-0181-6, Knowledge and Information Systems with permission. All rights reserved.

Inferences can also be performed according to some *problem-solving methods* (PSMs) such as *cover-and-differentiate* [Eshelman, 1988] or *propose-and-revise* [Marcus, 1988b, Leo et al., 1994]. These were originally expected to apply only to heuristic classification (diagnostics) and planning tasks [Bylander and Chandrasekaran, 1987, Marcus, 1988a]. Later, it was acknowledged that PSMs may have variable forms and that the relationship between tasks (or task families) and PSMs is not at all one-to-one [O'Hara and Shadbolt, 1993]. In other words, a task can be performed in many different ways.

In PSMs, much power can be gained by understanding the roles that domain knowledge plays in problem solving [Marcus, 1988a].

Different PSMs use different numbers of relation types and knowledge roles. They also define a pattern of using different knowledge roles in inference. For this reason, different PSMs require different COs in SOOKAT, conducting different patterns of message sending. The PSMs also determine the form of dependency graphs, i.e. the types of dependencies required.

18.4.1 Control object for cover-and-differentiate

The cover-and-differentiate PSM, originally used for fault detection [Eshelman, 1988], requires only a single type of dependency. The implementation of the cover-and-differentiate PSM can use the same CO as forward and backward chaining, as a similar dependency graph is used, and the PSM is an extension of forward chaining.

However, a separate CO with two sub-COs for the knowledge roles *covering knowledge* and *differentiating knowledge* can also be used. This CO follows the OO principle of modularity, as well as the general SOOKAT policy of generating COs.

Example 17. *The cover-and-differentiate PSM is almost directly applicable to mineral classification. The symptoms used as input are replaced with attribute values of a mineral sample, and the possible faults causing the symptoms are replaced with minerals that the sample may present.*

In SOOKAT, the main CO maintains a list of remaining solution concept class alternatives. The CO for the cover-and-differentiate PSM uses two

subordinate COs, one for covering and one for differentiating. The cover CO leaves in the list only the concepts fulfilling the current differentiating criterion (removing from the list items not fulfilling it). The differentiate CO finds a new differentiating criterion (requirement), finding a new criterion for differentiation through the attributes of the application instance. Differentiating criteria are needed as long as the list contains more than one remaining concept.

Example 18. *As no restrictions are originally set, the alternative list is, in the first cover phase, set to contain all concepts of the domain, i.e. Olivine and Pyroxene. The first differentiation criterion based on the attribute value of mineral#23 is*
"Mineral.colour is green."

Both candidates fulfil the criterion, so a new one is suggested:
"Mineral.crystal-shape is octagonal."
During the next cover phase, the concept Pyroxene is removed, as it does not fulfil the criterion. The only remaining concept, Olivine, is the solution.

18.4.2 Control object for propose-and-revise

There are three knowledge roles that knowledge can play in the PSM [Marcus, 1988b], *propose a design extension*, *identify a constraint*, and *propose a fix*. The dependency graph acquired should contain three types of dependencies. The CO for propose-and-revise has two sub-COs. The propose CO proposes both extensions of the KB and potential fixes, and the revise CO proposes a revision of the KB.

Example 19. *The propose-and-revise PSM can be used to solve the mineral classification problem. Minerals will, in practice, be gone through one by one.*

The PSM will be applied in SOOKAT as follows: Alternative Concepts are the hypotheses (identified one by one), and revision is performed by proposing a new Concept as a fix for a constraint violation.

Example 20. *In the mineral classification application, the propose CO first proposes both the KB extension*

"Mineral.classification is Pyroxene"

and the potential fix

"Mineral.classification is Olivine."

As the value of the attribute crystal-shape of Pyroxene contradicts that of the sample, the revise CO proposes the KB revision

"Mineral.classification is Olivine."

It can be noticed that the values of the attributes colour and crystal-shape are 'green' and 'octagonal' both in mineral#23 and Olivine. As the solution has been found, there is no need to propose a new mineral.

18.4.3 Other control objects

A mechanism for importing descriptions of PSMs from outside, e.g. in a similar fashion as in a broker selecting a suitable PSM from a library on the Internet [Benjamins et al., 1999], is under development. Descriptions are expected to be written by PSM providers (library holders) in the Unified Problem-solving Method description Language (UPML) [Fensel et al., 1999, Fensel and Motta, 2003].

A mechanism for generating COs based on the descriptions imported is also under development.

18.5 Reasoning in Sisyphus III rock classification

The rules to classify rock samples are

- those to assign a value for the attribute 'rocktype' in an unknown Rock instance according to the attributes, such as 'Rock.olivines', 'Rock.quartz' and 'Rock.silica', each having the amount of the corresponding mineral as the value,

- those to assign values for mineral attributes in a Rock instance according either to the areas occupied by different minerals or amounts of different minerals in the rock sample and

- those to assign values for mineral attributes in a Rock instance accord-

ing to chemical analysis of the rock sample.

18.5.1 Inference in Sisyphus III according to forward and backward chaining

Conclusion attributes with empty values (section 17.2) in Rock and Mineral application instances can be inferred, as in the mineral classification application, according to backward or forward chaining, using depth-first or breadth-first search, except that

- alternative solutions include 16 rocks instead of two minerals, and
- recognizing rocks requires recognition of minerals included and their relative proportions.

Rules for reasoning a value for the 'rocktype' attribute of a Rock sample can be roughly classified to the following groups:

1. The rules to infer the value of 'Rock.rocktype' from values of other Rock attributes,
2. the rules to infer the values of Rock attributes mentioned in item 1 from aggregate attributes of Rock denoting minerals, as described in section 15.2,
3. the rules to infer the values of Rock attributes mentioned in item 1 from measurements and findings in Rock samples, and
4. other rules.

18.5.2 Inference in Sisyphus III according to protocols

Protocols for recognizing rocks can take different approaches. A useful protocol for classifying rocks in the Sisyphus III application goes through the attributes of the rock samples and igneous minerals.

18.5.3 Inference in Sisyphus III according to problem-solving methods

In the Sisyphus III rock classification example, an acceptable choice is to use the cover-and-differentiate method as a problem-solving method (PSM) to classify rock samples. This is due to that rock types form the possible solutions based on which the solution is reasoned. Here attribute values replace symptoms used in the original PSM description [Kahn, 1988].

18.5.4 Generating explanations in Sisyphus III

Step-by-step explanations are quite complex to produce. Explanations are generated based on

- time stamps,
- analysis descriptions of inferences, and
- attribute values of concept instances.

18.6 Reasoning in dietary management of MS

Classes BodyInstance and DietInstance are both subclasses of class ConceptInstance. They both have attributes for components of personal diets.

18.6.1 Inference in dietary management of MS according to protocols

Again, a protocol means a documented series of reasoning stages (i.e. inference steps), that are based on some premise values, after which an acceptable result is reached [Ericsson and Simon, 1984].

Protocols for planning dietary regimens can take different approaches. A useful protocol for planning diets in the dietary management application goes through the attribute combinations of the dietary ingredients and medical or preferential restrictions (allergies, disease-related sensitivity, or dislikings).

18.6.2 Inference in dietary management of MS according to problem-solving methods

In the dietary management example, an acceptable choice is to use the propose-and-revise (PAR) method as a PSM to develop dietary regimens. This is due to that the method supports also planning.

The PAR method includes evaluation of possible solutions and selection of the most suitable one [Marcus, 1988a].

Evaluation of a solution (a dietary regimen) can be done using a formula, to be developed, referring to values of concept instance attributes

- vitamin B12,
- vitamin D,
- antioxidants,
- lactic acid, and
- fresh fruit and vegetables.

Amounts required are explained in chapter 9.

Example 21. *Three people with MS, having three bodies with MS, are considered.*

(a) BodyInstance#1 has high intake of saturated fats (more than 15% of saturated fatty acids, see section 9.3), has low intake of vitamin D, vitamin B12, Ω6 and Ω6 fatty acids and antioxidants, little excercise, and as a consequence runs the risk of getting severe longlasting symptoms often.

(b) BodyInstance#2 has low intake of saturated fats, high intake of vitamin D, high intake of vitamin B12, does intensive excercise, and has consequently good condition and little symptoms seldom if ever.

(c) BodyInstance#3 has infections and optimal diet, similar to that of BodyInstance#2, infections, and temporarily bad condition as a consequence.

18.6.3 Generating explanations in dietary management of MS

Step-by-step explanations are again generated based on

- time stamps,

- analysis descriptions of inferences, and
- attribute values of concept instances.

Part VII

SUMMARY AND DISCUSSION

19. Summary

19.1 Contributions

The methodology SeSKA (seamless structured knowledge acquisition) has been developed to reduce disintegration.

SOOKAT (structured object-oriented knowledge acquisition tool) supports SeSKA. It has manually been used to build a knowledge base (KB) for dietary management of multiple sclerosis (MS) [Parpola, 1998]. A SOOKAT KB for individual dietary planning is under development.

SOOKAT is continuously at a development stage. The Sisyphus III rock classification problem [Shadbolt et al., 1996] has been used throughout development of SOOKAT for testing different features.

19.2 Related work

Work concerning metaobjects and metalevel in knowledge acquisition includes the following:

Protégé-2000 [Fridman Noy et al., 2000] makes use of a metaobject protocol [Kiczales et al., 1991, Steele, 1990] to describe a model, e.g. the CommonKADS model of expertise [Schreiber et al., 2000]. This allows presentation of applications as instantiations of the model.

OIL (Ontology Inference Language) [Fensel et al., 2000] is a proposal for a joint standard for specifying and exchanging ontologies over the Internet. It is based on OKBC (Open Knowledge Base Connectivity), XOL (OntologyExchange Language), and RDF (Resource Description Framework). OIL distinguishes three layers in modelling

ontologies: the object level, the first metalevel, and the second metalevel. Several components are used in the structure: rule bases, classes and slots, types, slot constraints and inheritance. OIL is a frame-based system, using e.g. rule bases.

MODEL-ECS [Executable Conceptual Structures] [Lukose, 1995] also applies inferences in parallel with development.

Implementation of SOOKAT uses the Java programming language. Its object system has been enlarged to use two metaobject protocols,

- one for domain components, and
- one for components of reasoning.

Models forming the base of the KB (domain model and inference model) can be instantiated using the two metaobject protocols. Reasoning is perfomed according to different inference strategies using specific control objects (CO) and the instantiated models.

Contributions of this thesis include

- showing how an inference structure can be formed (via dependency graphs) based on inferential dependencies between components of a domain model,
- demonstrating one possible implementation of the SeSKA methodology, i.e. a structured set of object-oriented models,
- showing how instances can be created of domain model concepts and inference model roles in the value and execution models,
- showing how OO models created during the KA process can be used for performing inferences in several ways, and
- being able to directly utilize the KA models in inferences, making both building and maintenance of knowledge-based systems more efficient and more reliable.

Remaining trends of work include

- further investigations of inference in SOOKAT according to problem-solving methods (PSMs), using specific COs,
- construction of KBs for various kinds of applications,
- developing mechanisms for importing descriptions of PSMs, and exporting descriptions of COs and properties of instances over the Internet, using e.g. XML (extensible markup language),
- developing a mechanism for generating COs based on imported PSMs, and
- building embedded systems, using SOOKAT for reasoning.

References

[Azadbakht et al., 2002] Azadbakht, L., Kimiagar, M., and Zadeh, A. E. (2002). Correlation of fatty acid type intake with incidence of multiple sclerosis. In *ISSFAL 2002 Posters*, Montreal, Canada. International Society for the Study of Fatty Acids and Lipids. http://www.issfal.org.uk/posters-2.htm.

[Bates, 1990] Bates, D. (1990). Dietary lipids and multiple sclerosis. *Uppsala Journal of Medical Sciences Supplement*, 48:173–187.

[Benjamins et al., 1999] Benjamins, R., Wielinga, B., Wielemaker, J., and Fensel, D. (1999). Towards brokering problem-solving knowledge on the Internet. In Fensel, D. and Studer, R., editors, *Proceedings of EKAW99, 11th European Workshop on Knowledge Acquisition, Modeling and Management. Dagstuhl Castle, Germany, May 26 - 29, 1999*, pages 33–48. Springer-Verlag.

[Bratko, 1986] Bratko, I. (1986). *Prolog Programming for Artificial Intelligence*. Addison-Wesley Publishing Company, Wokingham, England.

[Brickley and Guha, 2004] Brickley, D. and Guha, R. (2004). RDF vocabulary description language 1.0: RDF schema. W3C recommendation, World Wide Web Consortium (W3C). http://www.w3.org/TR/rdf-schema.

[Bylander and Chandrasekaran, 1987] Bylander, T. and Chandrasekaran, B. (1987). Tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition. *Int. J. Man-Machine Studies*, 26:231–243. Also pp. 65-77 in [Gaines and Boose, 1988].

[Chiba, 1996] Chiba, S. (1996). *A Study of a Compile-time Metaobject Protocol*. PhD thesis, The University of Tokyo.

[Clausen et al., 1988] Clausen, J., Jensen, G., and Nielsen, S. (1988). Selenium in chronic neurologic diseases. multiple sclerosis and batten's disease. *Biol Trace Elem Res.*, 15:179–203.

[Cunnane et al., 1989] Cunnane, S., Ho, S., Dore-Duffy, P., Ells, K., and Horrobin, D. (1989). Essential fatty acid and lipid profiles in plasma and erythrocytes in patients with multiple sclerosis. *Am J Clin Nutr*, 50(4):801–6.

[de Hoog et al., 1994] de Hoog, R., Martil, R., Wielinga, B., Taylor, R., Bright, C., and van de Velde, W. (1994). The commonnkads model set. Technical Report KADS-II//M1 /DM1.1b/UvA/18/6.0/FINAL. DM1.1c., Esprit.

[Ericsson and Simon, 1984] Ericsson, K. and Simon, H. (1984). *Protocol Analysis*. MIT Press, Cambridge, MA, USA.

[Eshelman, 1988] Eshelman, L. (1988). MOLE: A knowledge-acquisition tool for cover-and-differentiate systems. In [Marcus, 1988a], pages 37–80.

[Eshelman et al., 1987] Eshelman, L., Ehret, D., McDermott, J., and Tan, M. (1987). MOLE: A tenacious knowledge acquisition tool. *Int. J. Man-Mach. St*, 26:41–54.

[FAO and WHO, 1994] FAO and WHO (1994). Fats and oils in human nutrition. FAO food and nutrition paper 57. Report of a joint expert consultation, Rome Oct 1993, Food and Agriculture Organization of the United Nations and the World Health Organization. http://www.fao.org/docrep/V4700E/V4700E00.htm.

[Fensel et al., 1999] Fensel, D., Benjamins, V., Decker, S., Gaspari, M., Groenboom, R., Grosso, W., Musen, M., Motta, E., Plaza, E., Schreiber, A. T., Studer, R., and Wielinga, B. (1999). The component model of UPML in a nutshell. In *Proceedings of the first working IFIP connference on software architecture (WICSA1),San Antonio, Texas, USA 1999*.

[Fensel and Motta, 2003] Fensel, D. and Motta, E. (2003). The unified problem–solving method development language (UPML). *Knowledge And Information Systems (KAIS)*, 5(1):83–131.

[Fensel et al., 2000] Fensel, D., van Harmelen, F., Decker, S., Erdmann, M., and Klein, M. (2000). OIL in a nutshell. In Dieng, R. and Corby, O., editors, *Knowledge Engineering and Knowledge Management. Methods, Models and Tools. 12th International Conference, EKAW 2000. Juan-les-Pins, France, October 2-6, 2000*.

[Fisher et al., 1987] Fisher, M., Johnson, M., Natale, A., and Levine, P. (1987). Linoleic acid levels in white blood cells, platelets, and serum of multiple sclerosis patients. *Acta Neurol Scand*, 76(4):293–326 or 241–245.

[Fitzgerald et al., 1987] Fitzgerald, G., Harbige, L., Forti, A., and Crawford, M. (1987). The effect of nutritional counselling on diet and plasma efa status in multiple sclerosis patients over 3 years. *Hum Nutr Appl Nutr*, 41(5):297–310.

[Food and Nutrition Board and Institute of Medicine, 2002] Food and Nutrition Board and Institute of Medicine (2002). *Dietary Reference Intakes for Energy, Carbohydrate, Fiber, Fat, Fatty Acids, Cholesterol, Protein, and Amino Acids (Macronutrients)*. The National Academic Press, United States.

[Foote and Johnson, 1989] Foote, B. and Johnson, R. E. (1989). Reflective facilities in smalltalk-80. In Meyrowitz, N., editor, *Conference on Object-Oriented Programming: Systems, Languages, and Applications (OOPSLA '89), October 1-6, 1989, New Orleans, Louisiana, Proceedings. SIGPLAN Notices 24(10)*, pages 327–335.

[Forgy, 1981] Forgy, C. (1981). OPS5 user's manual. Report CMU-CS-81-135, Carnegie-Mellon University, Department of Computer Science.

[Fridman Noy et al., 2000] Fridman Noy, N., Ferguson, R. W., and Musen, M. A. (2000). The knowledge model of Protégé-2000: Combining interoperabilithy and flexibity. In Dieng, R. and Corby, O., editors, *Knowledge Engineering and Knowledge Management. Methods, Models and Tools. 12th International Conference, EKAW 2000. Juan-les-Pins, France, October 2-6, 2000*. Springer-Verlag.

[Gaines and Boose, 1988] Gaines, B. and Boose, J. (1988). *Knowledge Acquisition for Knowledge-Based Systems*, volume 1. Academic Press, London, England.

[Gallai et al., 1995] Gallai, V., Sarchielli, P., Trequattrini, A., Franceschini, M., Floridi, A., Firenze, C., Alberti, A., Benedetto Di, D., and Stragliotto, E. (1995). Cytokine secretion and eicosanoid production in the peripheral blood mononuclear cells of ms patients undergoing dietary supplementation with n-3 polyunsaturated fatty acids. *J. Neuroimmunol*, 56(2):143–53.

[Goldberg, 1984] Goldberg, A. (1984). *Smalltalk-80: The interactive programming environment*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Grosso et al., 1999] Grosso, W., Eriksson, H., Feergerson, R., Gennari, J., Tu, S., and Musen, M. (1999). Knowledge modelling at the millenium. the design and evolution of Protégé-2000. In *12th Banff Workshop on Knowledge Acquisition, Modelling and Management, Banff, Alberta, Canada, 1999*.

[Harmon, 1991] Harmon, P. (1991). A brief overview of software methodologies. *Intelligent Software Strategies, the Monthly Newsletter on Expert Systems, OOP, CASE, Neural Networks and Natural Language*, 7(1).

[Heninger, 1980] Heninger, K. (1980). Specifying software requirements for complex systems. new techniques and their application. *IEEE Transactions in Software Engineering*, 6(1):2–13.

[Hesketh and Barrett, 1989] Hesketh, P. and Barrett, T., editors (1989). *An Introduction to the KADS Methodology*. STC Technology Ltd., Harlow, UK. ESPRIT Project P1098, Deliverable M1.

[Hutter, 1993] Hutter, C. (1993). On the causes of multiple sclerosis. *Med Hypotheses*, 41(2):93–6.

[Hyvönen and Koivistoinen, 1994] Hyvönen, L. and Koivistoinen, P. (1994). Fatty acid analysis, tag equivalents as net fat value, and nutritional attributes of fish and fish products. *J Food Comp Anal*, (7):44–58.

[Hyvönen et al., 1993] Hyvönen, L., Lampi, A.-M., Varo, P., and Koivistoinen, P. (1993). Fatty acid analysis, tag equivalents as net fat value, and nutritional attributes of commercial fats and oils. *J Food Comp Anal*, (6):24–40.

[Jacobson et al., 1992] Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G. (1992). *Object-Oriented Software Engineering, A Use Case Driven Approach*. Addison-Wesley, Reading, Massachusetts, USA.

[Jarke et al., 2002] Jarke, M., Jeusfeld, M. A., and Quix, C. (2002). *ConceptBase V5.2.3 User Manual*. RWTH Aachen, Germany. http://www-i5.informatik.rwth-aachen.de/CBdoc/userManual/.

[Java Data Objects Expert Group, 2003] Java Data Objects Expert Group (2003). Java data objects jsr12, version 1.0.1. Specification JSR-000012, Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, CA 95054, USA.

[Kahn, 1988] Kahn, G. (1988). More: From observing knowledge engineers to automating knowledge acquisition. In [Marcus, 1988a], pages 7–35.

[Kahn et al., 1985] Kahn, G., Nowlan, S., and McDermott, J. (1985). More: an intelligent knowledge acquisition tool. In *Ninth International Conference on Artificial Intelligence*.

[Kaiser, 1993] Kaiser, G. E. (1993). MARVEL 3: 1: A multi-user software development environment. In *ILPS*, pages 36–39.

[Kalueff et al., 2004] Kalueff, A., Eremin, K., and Tuohimaa, P. (2004). Mechanisms of neuroprotective action of vitamin d(3). *Biochemistry (Mosc)*, 69(7):738–41.

[Kiczales et al., 1991] Kiczales, G., des Riviers, J., and Bobrow, D. (1991). *The Art of the Metaobject Protocol*. MIT Press, Cambridge, MA, USA.

[Klinker, 1989] Klinker, G. (1989). A framework for knowledge acquisition. In Boose, J., Gaines, B., and Ganascia, J., editors, *Third European Workshop of Knowledge Acquisition for Knowledge-Based Systems*, pages 102–116.

[Kontio, 1991] Kontio, J. (1991). Matias: Development and maintenance of a large but well-defined expert systems with applications. *Expert Systems with Applications*, 3(2):241–248.

[KTL Nutrition Unit, 2004] KTL Nutrition Unit (2004). Fineli – finnish food composition database. Web interface for search on the database Release 4, National Public Health Institute of Finland. http://www.fineli.fi/.

[Leo et al., 1994] Leo, P., Sleeman, D., and Tsinakos, A. (1994). S-SALT, a problem solver plus; knowledge acquisition tool which additionally can refine its knowledge base. In *proc. of EKAW-94, the 8th European Knowledge Acquisition Workshop. Hoegaarden, Belgium, Artificial Intelligence Laboratory of the Vrije Universiteit Brussel*.
http://arti.vub.ac.be/ ekaw/welcome.html.

[Lindemann et al., 2000] Lindemann, T., Prange, A., Dannecker, W., and Neidhart, B. (2000). Stability studies of arsenic, selenium, antimony and tellurium species in water, urine, fish and soil extracts using HPLC/ICP-MS. *Fresenius' J. Anal. Chem.*, 368:214–220.

[Linster, 1992] Linster, M.and Musen, M. (1992). Use of kads to create a conceptual model of the oncocin task. *Knowledge Acquisition*, 4(4):55–87.

[Lukose, 1995] Lukose, D. (1995). Using executable conceptual structures for modelling expertise. In *Paper 8 in 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop. Banff, Alberta, Canada, February 26 - March 3, 1995*.

[Marcus, 1988a] Marcus, S., editor (1988a). *Automating Knowledge Acquisition for Expert Systems*. Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston.

[Marcus, 1988b] Marcus, S. (1988b). Salt: A knowledge-acquisition tool for propose-and-revise systems. In [Marcus, 1988a], pages 81–123.

[Marshall, 1991] Marshall, B.and Marshall, H. (1991). Lipids and neurological diseases. *Med Hypotheses*, 34:272–274.

[Mayer, 1991] Mayer, M. (1991). Linoleic-acid-dependent slowing of erythrocyte sedimentation in multiple sclerosis. *Prostaglandins Leukot Essent Fatty Acids*, 44(4):57–8.

[McDermott, 1988] McDermott, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In [Marcus, 1988a], pages 225–256.

[McGuinness and van Harmelen, 2004] McGuinness, D. L. and van Harmelen, F. (2004). OWL web ontology language overview. W3C recommendation, World Wide Web Consortium (W3C), Massachusetts Institute of Technology (MIT), Computer Science and Artificial Intelligence Laboratory (CSAIL), 32 Vassar Street, Room 32-G515, Cambridge, MA 02139, USA. http://www.w3c.org/TR/owl-features/.

[Miller et al., 2005] Miller, A., M., K., Almog, R., and Galboiz, Y. (2005). Vitamin B12, demyelination, remyelination and repair in multiple sclerosis. *Journal of the Neurological Sciences*, 233(1-2):93–97.

[Motta et al., 1988] Motta, E., Rajan, T., and Eisenstadt, M. (1988). A methodology and tool for knowledge acquisition in keats-2. In *3rd AAAI- Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop. Banff, Canada, 6-11 November,1988*, pages 21/1–20.

[Munger et al., 2004] Munger, K. et al. (2004). Vitamin D intake and incidence of multiple sclerosis. *Neurology*, 62:60–65.

[Musen, 1989a] Musen, M. (1989a). Conceptual models of interactive knowledge acquisition tools. *Knowledge Acquisition*, pages 73–88.

[Musen, 1989b] Musen, M. (1989b). Knowledge acquisition at the metalevel: Creation of custom-tailored knowledge acquisition tools. *ACM SIGART Newsletter*, 108:45–55.

[Musen et al., 1988] Musen, M., Fagan, L., Combs, D., and Shortcliffe, E. (1988). Use of a domain model to drive an interactive knowledge-editing tool. In *Knowledge Acquisition Tools for Expert Systems*, pages 257–273. Academic Press.

[Navarro and Segura, 1988] Navarro, X. and Segura, R. (1988). Plasma lipids and their fatty acid composition in multiple sclerosis. *Acta Neurol. Scand.*, 78:152–157.

[Navarro and Segura, 1989] Navarro, X. and Segura, R. (1989). Red blood cell fatty acids in multiple sclerosis. *Acta Neurol Scand.*, 79:32–37.

[Neu, 1985] Neu, I. (1985). Metabolic aspects of multiple sclerosis stoffwechselaspekte der multiplen sklerose. *Wien Med Wochschr.*, 135(1-2):20–2.

[Nightingale et al., 1990] Nightingale, S., Woo, E., Smith, A., French, J., Gale, M., Sinclair, H., Bates, D., and Shaw, D. (1990). Red blood cell and adipose tissue fatty acids in mild inactive multiple sclerosis. *Acta Neurol. Scand.*, 82(1):43–50.

[Noy and Klein, 2004] Noy, N. and Klein, M. (2004). Ontology evolution: not the same as schema evolution. *Knowledge And Information Systems*, 6(4).

[Object Management Group, Inc., 2002] Object Management Group, Inc. (2002). *Meta Object Facility (MOF) Specification, version 1.4*. http://www.omg.org/cgi-bin/doc?formal/2002-04-03.

[Object Management Group, Inc., 2003] Object Management Group, Inc. (2003). *XML Metadata Interchange (XMI) Specification, version 2.0*. http://www.omg.org/cgi-bin/doc?formal/2003-05-02.

[O'Hara and Shadbolt, 1993] O'Hara, K. and Shadbolt, N. (1993). Locating generic tasks. *Knowledge Acquisition*, 5(5):449–481.

[Parpola, 1998] Parpola, P. (1998). Seamless development of structured knowledge bases. In B Gaines, M. M., editor, *Proceedings of KAW98, Eleventh Workshop on Knowledge Acquisition, Modeling and Management. Banff, Alberta, Canada, 18th-23rd April, 1998*. University of Calgary. http://ksi.cpsc.ucalgary.ca/KAW/KAW98/parpola/.

[Parpola, 1999a] Parpola, P. (1999a). Applying SeSKA to Sisyphus III. In Fensel, D. and Studer, R., editors, *Knowledge Acquisition, Modeling and Management. Proceedings of , 11th European Workshop, EKAW '99, Dagstuhl Castle, Germany, May 26 - 29, 1999*, number 1621 in Lecture Notes in Artificial Intelligence. Springer-Verlag.

[Parpola, 1999b] Parpola, P. (1999b). Development and inference in integrated OO models. In Mohammadian, M., editor, *CIMCA'99 - The international conference on computational intelligence for modelling, control and automation. Vienna, Austria, 17-19 February, 1999*. IOS Press.

[Parpola, 2000] Parpola, P. (2000). Managing terminology using statistical analysis, ontologies and a graphical KA tool. In Aussenac-Gilles, N., Biebow, B., and Szulman, S., editors, *Proceedings of the International Workshop on Ontologies and Texts during EKAW2000, 12th European Workshop on Knowledge Engineering and Knowledge Management, Juan-les-Pins, French Riviera, October 2–6, 2000*. French group of interest TIA.

[Parpola, 2001] Parpola, P. (2001). Integration of development, maintenance and use of knowledge bases. In *Proceedings of the Workshop on Knowledge Management and Organizational Memory during IJCAI'01, International Joint Conference on Artificial Intelligence, Seattle, Washington, USA, August 4-10, 2001*. IJCAI consortium.

[Parpola, 2002] Parpola, P. (2002). Integration of development, maintenance and use of knowledge bases (revised version). In Dieng, R. and Matta, N., editors, *Knowledge Management and Organizational Memory*, pages 41–50. Kluwer Academic Publishers.

[Parpola, 2005] Parpola, P. (2005). Inference in the SOOKAT object-oriented knowledge acquisition tool. *Knowledge and Information Systems*, (8):310–329.

[Reynolds et al., 1992] Reynolds, E., Bottiglieri, T., Laundy, M., Crellin, R., and Kirker, S. (1992). Vitamin b12 metabolism in multiple sclerosis. *Archives of Neurology*, 49(6):649–652.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., and F. Eddy, W. L. (1991). *Object-Oriented Modeling and Design*. Prentice-Hall. Englewood Cliffs, New Jersey.

[Schreiber et al., 1999] Schreiber, A., Akkermans, J. M., Anjewierden, A. A., de Hoog, R., Shadbolt, N. R., Van de velde, W., and Wielinga, B. (1999). *Knowledge Enginering and Management: The CommonKADS Methodology*. MIT Press, Cambridge, MA.

[Schreiber et al., 1989] Schreiber, G., Bredeweg, B., de Greef, P., Terpstra, P., Wielinga, B., Brumet, E., Simonin, N., and Wallyn, A. (1989). A KADS approach to KBS design. ESPRIT Project 1098, deliverable B6 UvA-B6-PR-010, University of Amsterdam & Cap Sogeti Innovation.

[Schreiber et al., 2000] Schreiber, G., Crubézy, M., and Musen, M. (2000). A case-study in using Protégé-2000 as a tool for CommonKADS. In Dieng, R. and Corby, O., editors, *EKAW2000, 12th European Workshop on Knowledge Engineering and Knowledge Management, Juan-les-Pins,French Riviera, October 2–6, 2000*. Springer-Verlag.

[Schreiber et al., 1994] Schreiber, G., Wielinga, B., de Hoog, R., Akkermans, H., and van de Velde, W. (1994). CommonKADS, a comprehensive methodology for kbs development. *IEEE Expert*, 9(6):28–37.

[Shadbolt et al., 1996] Shadbolt, N., Crow, L., Tennison, J., and Cupit, J. (1996). Sisyphus iii phase 1 release. http://www.psychology.nottingham.ac.uk/research/ai/sisyphus/SisIII-page.html.

[Sinclair, 1984] Sinclair, H. (1984). Essential fatty acids in perspective. human nutrition. *Clin. Nutr.*, 38(4):245–260.

[Sowa, 1984] Sowa, J. (1984). *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, Reading, Massachusets.

[Steele, 1990] Steele, G. L. (1990). *Common Lisp, the Language, Second Edition*. Digital Press, USA.

[Sun Microsystems, Inc., 2001] Sun Microsystems, Inc. (2001). *Java 2 SDK, Standard Edition, Documentation Version 1.3.1*. Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, CA 95054, USA. http://java.sun.com/j2se/1.3/docs/.

[Swank, 1991] Swank, R. (1991). Multiple sclerosis: fat-oil relationship. *Nutrition*, 7:368–376.

[Toshniwal and Zarling, 1992] Toshniwal, P. and Zarling, E. (1992). Evidence for increased lipid peroxidation in multiple sclerosis. *Neurochem Res.*, 17:205–207.

[USDA Nutrient Data Laboratory, 2004] USDA Nutrient Data Laboratory (2004). Usda national nutrient database for standard reference. Web interface for search on the database Release 17, United States Department of Agriculture. http://www.nal.usda.gov/fnic/foodcomp/.

[van der Mei et al., 2003] van der Mei, I. et al. (2003). Past exposure to sun, skin phenotype, and risk on multiple sclerosis: case-control study. *British Medical Journal*, 327(7410):316.

[VanAmerongen et al., 2004] VanAmerongen, B. M., Dijkstra, C. D., Lips, P., and Polman, C. H. (2004). Multiple sclerosis and vitamin D: an update. *Eur J Clin Nutr.*, 58(8):1095–109.

[Wetter, 1990] Wetter, T. (1990). First-order logic foundations of the KADS conceptual model. In Wielinga, B. et al., editors, *Current Trends in Knowledge Acquisition*.

[Wielinga and Breuker, 1986] Wielinga, B. and Breuker, J. (1986). Models of expertise. In *ECAI '86, Seventh European Conference on Artificial Intelligence, July 1986. Brighton, UK*.

[X3J20 Workgroup, 1998] X3J20 Workgroup (1998). ANSI Smalltalk. Standard ANSI/NCITS 319-1998, ANSI.

[Zamaria, 2004] Zamaria, N. (2004). Alteration of polyunsaturated fatty acid status and metabolism in health and disease. *Reproduction Nutrition Development*, 44(3):273–282.

Lic.Phil. Päivikki Parpola (1965-2015) presents in this research report the SeSKA (seamless structured knowledge acquisition) methodology, integrating phases of knowledge acquisition (KA) through seamless transformations. This attacks the problem of disintegration, or the gap between phases. The methodology is accompanied by presentation of the SOOKAT (structured object-oriented knowledge acquisition) tool supporting it. SeSKA and SOOKAT extend the KA process to constructing knowledge bases by instantiating a series of models for inferencing. The models are object-oriented and they are constructed in SOOKAT utilizing meta-object protocols. The approach is novel, although the main contribution was made before year 2006. Mechanisms for importing problem-solving methods (PSMs) over the Internet, as well as for generating specific control objects (COs) for them, remain open to further development.

Aalto-ST 13/2015

ISBN 978-952-60-6460-4 (printed)
ISBN 978-952-60-6461-1 (pdf)
ISSN-L 1799-4896
ISSN 1799-4896 (printed)
ISSN 1799-490X (pdf)

Aalto University
School of Science
Department of Computer Science
www.aalto.fi

BUSINESS + ECONOMY

ART + DESIGN + ARCHITECTURE

SCIENCE + TECHNOLOGY

CROSSOVER

DOCTORAL DISSERTATIONS